THE EXPERT'S VOICE® IN WEB DEVELOPMENT

jQuery

ДЛЯ ПРОФЕССИОНАЛОВ

Освойте самые эффективные приемы работы с библиотекой jQuery

Адам Фримен

Apress®

jQuery
ДЛЯ ПРОФЕССИОНАЛОВ

Pro jQuery

Adam Freeman

Apress®

jQuery

ДЛЯ ПРОФЕССИОНАЛОВ

Адам Фримен

Москва • Санкт-Петербург • Киев
2013

ББК 32.973.26-018.2.75
Ф88
УДК 681.3.07

Издательский дом "Вильямс"
Главный редактор *С.Н. Тригуб*
Зав. редакцией *В.Р. Гинзбург*
Перевод с английского и редакция канд. хим. наук *А.Г. Гузикевича*
По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Фримен, Адам.
Ф88 jQuery для профессионалов. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2013. — 960 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1799-7 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание
Адам Фримен
jQuery для профессионалов

Литературный редактор *Л.Н. Красножон*
Верстка *М.А. Удалов*
Художественный редактор *Е.П. Дынник*
Корректор *Л.А. Гордиенко*

Подписано в печать 03.11.2012. Формат 70x100/16
Гарнитура Times. Печать офсетная
Усл. печ. л. 77,4. Уч.-изд. л. 48,22
Тираж 1500 экз. Заказ № 3393

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9-я линия, 12/28

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1799-7 (рус.)
ISBN 978-1-4302-4095-2 (англ.)

Оглавление

Содержание

Об авторе

Адам Фримен — профессионал в области информационных технологий с опытом руководящей работы в целом ряде компаний. Свою карьеру завершил в должности технического директора в одном из международных банков. После выхода на пенсию все свое время посвящает написанию книг. Это четырнадцатая из его книг, посвященных передовым технологиям программирования.

О техническом рецензенте

Фабио Клаудио Ферракьяти (Fabio Claudio Ferracchiati) — старший консультант и ведущий аналитик-разработчик в итальянском филиале компании Brain Force (`www.brainforce.com`). Его специализация — технологии Microsoft. Фабио — обладатель сертификатов MCSD (Microsoft Certified Solution Developer) и MCAD (Microsoft Certified Application Developer) для .NET, а также MCP (Microsoft Certified Professional). За последние десять лет опубликовал множество статей в итальянских и международных журналах и выпустил более десяти книг, посвященных современным компьютерным технологиям.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info@williamspublishing.com
WWW: http://www.williamspublishing.com

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1
в Украине: 03150, Киев, а/я 152

часть I

Подготовка к работе

ГЛАВА 1

Подключение jQuery

Если вкратце охарактеризовать то, что делает библиотека jQuery[1], то это прозвучит довольно обыденно: данная библиотека позволяет изменять содержимое HTML-документов путем манипулирования объектами модели, создаваемой браузерами в процессе обработки HTML-кода (так называемые *DOM-манипуляции*, которые еще будут обсуждаться нами более подробно). Коль скоро вы читаете эти строки, то вам, наверное, уже приходилось манипулировать объектами DOM (Document Object Model — объектная модель документа) с помощью других библиотек JavaScript или встроенных функций API (Application Programming Interface — интерфейс программирования приложений), которые поддерживаются большинством современных браузеров, и данная книга понадобилась вам для того, чтобы делать это еще лучше.

Однако библиотека jQuery работает не просто лучше. Она превращает манипулирование объектами DOM в увлекательное занятие, временами напоминающее настоящее развлечение. В работе jQuery есть нечто настолько элегантное и притягательное, что задачи, решение которых обычно требует выполнения множества скучных рутинных операций, внезапно чрезвычайно упрощаются, так что, начав использовать jQuery, вы уже не сможете от этого отказаться. Что касается лично меня, то я использую jQuery в своих проектах по следующим причинам.

- Средства jQuery необычайно выразительны. Эта библиотека позволяет добиться гораздо большего при намного меньшем объеме кода, чем в случае использования программных DOM-интерфейсов браузеров.
- Методы jQuery применимы к целым группам элементов. Предлагаемый в DOM-модели стандартный подход, основанный на шаблоной цепочке действий "выбрать–повторить–изменить", больше не требуется. Следствием этого является уменьшение количества циклов `for` в коде, а значит, и снижение вероятности появления в нем ошибок.
- Библиотека jQuery справляется с различиями в реализации DOM в различных браузерах (проблемы кросс-браузерности). Например, меня не должна беспокоить мысль об особенностях поддержки того или иного средства, чем печально славится браузер Internet Explorer (IE). Достаточно всего лишь сформулировать jQuery свои пожелания, и библиотека самостоятельно обеспечит совместимость с конкретным браузером.
- Библиотека jQuery имеет открытый исходный код. Если принципы работы какого-либо средства для меня не совсем ясны или получаемый результат не совпадает с ожидаемым, я могу обратиться непосредственно к коду библиотеки на JavaScript и, если это необходимо, внести соответствующие изменения.

---

[1] Первая альфа-версия библиотеки jQuery была представлена ее автором Джоном Резигом на международной компьютерной конференции "BarCamp" в Нью-Йорке в 2006 г. — *Примеч. ред.*

Разумеется, в нашем мире нет ничего идеального, и в библиотеке имеется несколько "подводных камней", о чем мы еще будем говорить при более детальном обсуждении. И все же, несмотря на наличие отдельных недостатков, мне нравится работать с этой библиотекой. Я надеюсь, что вы также будете в восторге от нее и работа с ней доставит вам удовольствие. Библиотека jQuery обладает тем удивительным свойством, что задачи, которые обычно доставляют много хлопот в процессе разработки, решаются с ее помощью быстро и просто. Можно ли желать большего?

Библиотеки jQuery UI и jQuery Mobile

В данной книге рассматриваются также библиотеки *jQuery UI* и *jQuery Mobile*, которые представляют собой библиотеки элементов пользовательского интерфейса (UI) веб-приложений, реализованные поверх jQuery. Библиотека jQuery UI — это набор инструментальных средств, который предназначен для создания универсальных пользовательских интерфейсов и может применяться на любых устройствах, тогда как набор jQuery Mobile ориентирован на устройства, обладающие возможностями сенсорного ввода, такие как смартфоны или планшетные компьютеры.

Подключаемые модули jQuery

Подключаемые модули (плагины) jQuery расширяют функциональность базовой библиотеки. Некоторые подключаемые модули настолько эффективны и широко используются, что будут рассмотрены отдельно. Для jQuery разработано множество плагинов (хотя качество некоторых из них оставляет желать лучшего), и если какие-то из числа описанных в книге вам не понравятся, можно с уверенностью утверждать, что средства библиотеки позволят вам реализовать альтернативный вариант, который вас устроит.

Что необходимо знать читателю

Эта книга принесет максимальную пользу тем читателям, которые знакомы с основами веб-разработки, понимают принципы работы HTML и CSS и, в идеальном случае, имеют опыт работы с JavaScript. Для тех, кого кое-что из перечисленного заставляет чувствовать себя неуверенно, в главах 2–4 содержится дополнительный материал, который поможет вам освежить или пополнить свои знания в перечисленных областях. Разумеется, рассчитывать на то, что в указанных главах вы найдете подробные справочные сведения, охватывающие все без исключения элементы HTML или свойства CSS, не следует. Для рассмотрения HTML во всей его полноте в книге, посвященной jQuery, просто не хватило бы места.

Структура книги

Книга состоит из шести частей, каждая из которых охватывает несколько родственных тем.

Часть I. Подготовка к работе

В части I приведена информация, которая подготовит вас к чтению остальных частей. В эту часть входят текущая глава, а также главы, содержащие минимальный набор необходимых сведений по HTML, CSS и JavaScript. Программное обеспечение, которое понадобится вам для выполнения примеров, иллюстрирующих излагаемый материал, описано далее.

Часть II. Работа с jQuery

В части II, которая познакомит вас с библиотекой jQuery, мы начнем работать с базовым примером, постепенно наращивая его путем использования каждой из основных функциональных возможностей jQuery, таких как выбор элементов, манипулирование объектами DOM, обработка событий и спецэффекты.

Часть III. Работа с данными и Ajax

В части III описаны предлагаемые в jQuery методы для работы со встроенными и дистанционными данными. Здесь вы узнаете, как генерировать HTML-содержимое на основе данных, верифицировать данные, вводимые в веб-формы, и применять jQuery для выполнения асинхронных операций, в том числе с использованием возможностей Ajax.

Часть IV. Использование библиотеки jQuery UI

Библиотека jQuery UI — это одна из двух рассматриваемых в данной книге библиотек элементов пользовательского интерфейса. Реализованная поверх ядра библиотеки jQuery и интегрированная в нее, библиотека jQuery UI позволяет создавать функционально насыщенные интерфейсы для веб-приложений, характеризующиеся высокой степенью интерактивности.

Часть V. Использование библиотеки jQuery Mobile

Библиотека jQuery Mobile — это вторая из описанных в данной книге библиотек элементов пользовательского интерфейса. Она также реализована поверх jQuery и заимствует некоторые базовые возможности из jQuery UI, однако оптимизирована с учетом специфики интерфейсов приложений, выполняющихся на смартфонах и планшетах. Число доступных в jQuery Mobile виджетов пользовательского интерфейса меньше, чем в jQuery UI, но те из них, которые поддерживаются, оптимизированы для обеспечения интерактивного взаимодействия с устройствами посредством касаний и жестов и представления содержимого на экранах меньшего размера.

Часть VI. Дополнительные возможности

В заключительной части книги описаны некоторые возможности jQuery и jQuery UI, которые находят лишь ограниченное применение, но могут быть полезными в сложных проектах. Использование этих дополнительных возможностей требует более глубокого понимания HTML, CSS и средств самой библиотеки jQuery. Материал главы 35 будет более понятен тем читателям, которые имеют хотя бы общее представление о том, что такое асинхронное программирование.

Листинги примеров

В этой книге содержится *множество* примеров. Одна из приятных особенностей jQuery — возможность решения практически любой задачи несколькими способами, что открывает перед вами широкие перспективы для выработки собственного стиля работы с библиотекой. Многочисленные примеры раскрывают все разнообразие доступных альтернативных подходов. Примеров настолько много, что каждый из HTML-документов, с которыми вам придется работать, приводится в полном виде лишь один раз в начале каждой главы, иначе книга разрослась бы до невероятных размеров. Первый из включенных в каждую главу примеров будет представлять собой законченный HTML-документ наподобие того, который представлен в листинге 1.1.

Листинг 1.1. Пример законченного HTML-документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var labelElems =
                document.getElementsByTagName("label");
            var jq = $('img[src*=daffodil]');

            $('img:even').add('img[src*=primula]').add(jq)
                .add(labelElems).css("border", "thick double
                red");

        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
```

```
                </div>
            </div>
            <div id="row2"class="drow">
                <div class="dcell">
                    <img src="peony.png"/>
                    <label for="peony">Пионы:</label>
                    <input name="peony" value="0" required>
                </div>
                <div class="dcell">
                    <img src="primula.png"/>
                    <label for="primula">Примулы:</label>
                    <input name="primula" value="0" required>
                </div>
                <div class="dcell">
                    <img src="snowdrop.png"/>
                    <label for="snowdrop">Подснежники:</label>
                    <input name="snowdrop" value="0" required>
                </div>
            </div>
        </div>
    </div>
    <div id="buttonDiv">
        <button type="submit">Заказать</button>
    </div>
</form>
</body>
</html>
```

Этот листинг взят из главы 5. Не пытайтесь сейчас понять, как работает данный код. Просто знайте, что первый из приведенных в каждой главе примеров представляет собой завершенный HTML-документ, аналогичный только что приведенному. Почти все примеры строятся на основе одного и того же базового HTML-документа, отображающего простую веб-страницу цветочного магазина. Конечно, этот документ — не шедевр, но для наших целей его будет вполне достаточно. Он включает в себя все то, что может представлять для нас интерес с точки зрения изучения возможностей jQuery.

Для второго и всех последующих примеров в каждой главе будут обсуждаться лишь изменения, вносимые в исходный документ. Как правило, эти изменения будут касаться элемента `script`, содержащего код jQuery. Признаком того, что приводимый код является фрагментом, а не полным документом, будут служить многоточия в начале и конце кода, как показано в листинге 1.2.

Листинг 1.2. Неполный листинг

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $('label');

        // выбрать и изменить первый элемент
        jq.first().css("border", "thick double red");

        // выбрать и изменить последний элемент
        jq.last().css("border", "thick double green");

        // выбрать и изменить индексированный элемент
```

```
        jq.eq(2).css("border", "thick double black");
        jq.eq(-2).css("border", "thick double black");

    });
</script>
...
```

Это второй из приведенных в главе 5 примеров. В соответствии со сделанными замечаниями он представляет лишь элемент `script`, в котором выделены некоторые инструкции. Тем самым ваше внимание привлекается к той части примера, в которой используются изучаемые в данный момент возможности jQuery. Неполные листинги, подобные этому, отражают лишь изменения, которые вносятся в исходный документ, приведенный в начале соответствующей главы.

Каждый пример фокусируется на каком-то одном определенном средстве. Такой подход максимально упрощает освоение методов работы с jQuery. Однако при этом возрастает и риск того, что увидеть общую картину взаимодействия различных средств jQuery будет сложнее. Поэтому каждая часть книги завершается короткой главой, в которой код примера подвергается рефакторингу с целью включения в него материала, обсуждавшегося в предыдущих главах, и формирования у читателя цельного представления о доступных возможностях.

Исходные коды примеров

Чтобы не вводить вручную исходный код примеров в процессе их воспроизведения, можете загрузить его вместе со всеми вспомогательными ресурсами (включая изображения, библиотеки JavaScript и таблицы стилей CSS) на сайте Apress.com[2]. Делать это необязательно, однако так вам будет легче экспериментировать с примерами и вырезать нужные фрагменты кода для вставки в собственные проекты.

Совет. В то время как в большинстве листингов, приведенных в книге, отражаются лишь вносимые в код изменения, листинги, находящиеся на сайте книги, представляют завершенные HTML-документы, пригодные для непосредственной загрузки в браузер.

Программное обеспечение, необходимое для работы с книгой

Для выполнения приведенных в книге примеров вам понадобится описанное ниже программное обеспечение.

Библиотека jQuery

Прежде всего, вам понадобится библиотека jQuery, доступная для бесплатной загрузки на сайте `http://jquery.com`. В правой части главной страницы указанного сайта находится кнопка `Download`, щелчок на которой позволяет загрузить файл библиотеки, а с помощью расположенных над этой кнопкой переключателей можно выбрать, какой из двух доступных вариантов этого файла должен быть загружен: сжатый (`Production`) или несжатый (`Development`) (рис. 1.1).

---

[2] Файлы примеров продублированы также на сайте издательского дома "Вильямс" по адресу `http://www.williamspublishing.com/Books/978-5-8459-1799-7.html`. — *Примеч. ред.*

Рис. 1.1. Загрузка библиотеки jQuery

В данной книге мы будем использовать несжатый вариант файла библиотеки. О сути различий между двумя указанными вариантами файла, а также о том, как пользоваться библиотекой jQuery, говорится в главе 5.

Примечание. Процедуры загрузки и установки библиотек jQuery UI и jQuery Mobile описаны в главах 17 и 26.

HTML-редактор

К числу важнейших средств веб-разработки относятся текстовые редакторы, которые могут использоваться для создания HTML-документов. HTML-документ — это всего лишь текст, для работы с которым достаточно иметь самый простой текстовый редактор, однако существуют специализированные программы (и среди них есть много бесплатных), которые позволяют максимально упростить процесс разработки и добиться того, чтобы он протекал гладко и безболезненно[3].

Чаще всего я использую для этих целей программу Komodo Edit компании Active State. Она бесплатна, проста в применении и предлагает достаточно хорошую поддержку HTML, JavaScript и jQuery. С компанией Active State меня ничто не связывает, если не считать того, что я пользуюсь ее программным обеспечением. Чтобы загрузить программу Komodo Edit, посетите сайт `http://activestate.com`, на котором доступны версии этой программы для Windows, Mac и Linux.

---

[3] Кроме перечисленных автором текстовых редакторов, можно порекомендовать бесплатный текстовый редактор с открытым исходным кодом Notepad++, среди возможностей которого следует, в частности, отметить выбор языка пользовательского интерфейса, подсветку синтаксиса, сворачивание кода, поддержку регулярных выражений и удобные средства преобразования кодировок ANSI, UTF-8 и UCS-2. Программа доступна для загрузки по адресу `http://notepad-plus-plus.org/`. — *Примеч. ред.*

В качестве альтернативного варианта можно порекомендовать JsFiddle — популярный онлайновый редактор, поддерживающий работу с jQuery. Меня он не совсем устраивает (манера его структуризации противоречит моим привычкам разработчика), однако, судя по всему, он довольно гибок и обладает неплохими возможностями. Этот текстовый редактор доступен для бесплатной загрузки по адресу `http://jsfiddle.net`.

Веб-браузер

Для просмотра HTML-документов и тестирования кода jQuery и JavaScript вам понадобится веб-браузер. Мне нравится Google Chrome: он быстро работает и имеет простой интерфейс, а предоставляемые им инструменты разработки достаточно эффективны. Все снимки экрана, с которыми вы будете сталкиваться в книге (а их довольно много), получены именно с использованием браузера Google Chrome.

Из сказанного вовсе не следует, что вы должны использовать тот же браузер, что и я, однако желательно, чтобы выбранный вами веб-браузер был оснащен современными средствами разработки. Превосходный инструментарий для работы с JavaScript в браузере Mozilla Firefox предоставляется расширением Firebug, доступным для загрузки по адресу `http://getfirebug.com`.

Если по каким-либо причинам ни Chrome, ни Firefox вам не подходят, то следующий наилучший кандидат — это Internet Explorer (IE). Проблемы при работе с IE возникают у многих программистов, однако, исходя из своего опыта, могу сказать, что версия IE9 работает неплохо, а версия IE10 (которая на момент написания данной книги существовала только в виде бета-версии) кажется многообещающей. Предлагаемый в этой версии набор инструментов разработки не столь полон, как в браузерах Chrome или Firefox, но для целей книги его вполне достаточно.

Веб-сервер

Если вы захотите воспроизводить приводимые в книге примеры, вам потребуется веб-сервер, который браузер мог бы использовать для загрузки примеров HTML-документов вместе с соответствующими ресурсами (такими, как изображения или файлы JavaScript). Существует множество веб-серверов, большинство из которых предоставляется бесплатно и относится к категории программного обеспечения с открытым исходным кодом. Какой именно веб-сервер вы будете использовать, не имеет значения. Для этой книги я использовал веб-сервер Microsoft IIS 7.5, но лишь потому, что на моей машине уже была установлена и готова к работе операционная система Windows Server[4].

Node.js

Начиная с части III в дополнение к обычному веб-серверу мы будем использовать библиотеку Node.js, обеспечивающую серверную реализацию JavaScript на основе движка V8. В настоящее время Node.js пользуется огромной популярностью, но я использовал эту библиотеку по той простой причине, что она основана на JavaScript, и это избавляет от необходимости привлекать какие-либо отдельные

[4] Читателям, не имеющим готового серверного решения, можно порекомендовать кроссплатформенную сборку XAMPP, которая включает в себя веб-сервер Apache и доступна для бесплатной загрузки по адресу `http://www/apachefriends.org/en/xampp.html`. — *Примеч. ред.*

фреймворки для разработки веб-приложений. Вам не придется разбираться в деталях того, как работает библиотека Node.js, поскольку мы будем использовать ее по принципу "черного ящика" (однако на тот случай, если вам это интересно, в книге приведены некоторые серверные сценарии, дающие представление о том, что происходит на сервере).

Библиотеку Node.js можно загрузить по адресу `http://nodejs.org`. Она доступна в виде предварительно скомпилированного двоичного кода для Windows и в виде исходного кода, пригодного для использования на других платформах. В книге используется версия 0.5.9, хотя к тому времени, когда вы будете читать эти строки, она уже может быть заменена более новой версией. Однако и в этом случае можно ожидать, что никаких проблем с работой серверных сценариев у вас возникать не будет.

Настройка и тестирование сервера Node.js

Простейший способ тестирования Node.js — попытаться выполнить какой-либо несложный сценарий. Сохраните содержимое листинга 1.3 в файле с именем `NodeTest.js`. На моем компьютере он находится в той же папке, что и исполняемый двоичный код сервера Node.js.

Листинг 1.3. Тестовый сценарий для сервера Node.js

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
    console.log("Request: " + req.method + " to " +
        req.url);

    res.writeHead(200, "OK", {'Content-Type': 'text/html;
        charset=UTF-8'});
    res.write("<h1>Привет</h1>Node.js работает");
    res.end();

}).listen(80);
console.log("Ready on port 80");
```

Этот простой тестовый сценарий возвращает фрагмент HTML-кода в ответ на получение HTTP-запроса, использующего метод `GET`.

Примечание. Не волнуйтесь, если последнее предложение вам ни о чем не говорит. Чтобы использовать jQuery, вы вовсе не обязаны знать принципы работы HTTP и веб-сервера, тогда как краткое введение в HTML содержится в главе 2.

Чтобы протестировать фреймворк Node.js, запустите исполняемый файл, указав для него в качестве аргумента имя только что созданного файла. На своем компьютере, работающем под управлением Windows, я ввел в окне командной строки следующую команду.

```
node NodeTest.js
```

Проверьте, что все работает нормально, указав в адресной строке браузера адрес сервера, подключенного к порту 80 той машины, на которой выполняется

Node.js[5]. Картинка, которую вы увидите, должна быть аналогична той, которая представлена на рис. 1.2.

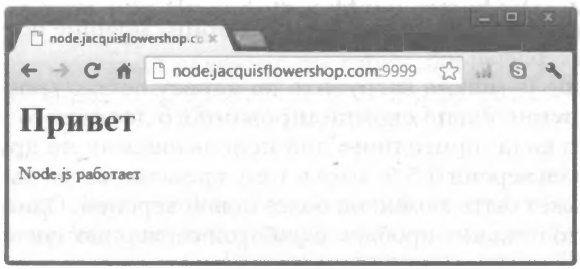


Рис. 1.2. Тестирование Node.js

В моем случае обычный веб-сервер и Node.js выполнялись на разных машинах, поэтому никаких проблем с использованием порта 80 не возникало. Если вы работаете только с одной машиной, используйте для веб-сервера порт 80 или измените сценарий `NodeTest.js`, задав в нем другой порт. Та часть сценария, в которой вы должны указать, какой именно порт следует использовать при тестировании, выделена в листинге 1.3 полужирным шрифтом[6].

Изображения для сайта

Во всех примерах книги используется набор графических изображений для сайта цветочного магазина. Эти изображения мне любезно предоставили многие люди, в том числе Хория Варлан, Дейвид Шорт, Танака Юйо, Мерви Эскелайнен и Алан Крейги.

Резюме

В этой главе мы вкратце познакомились с содержанием и структурой книги в целом и рассмотрели программное обеспечение, которое потребуется вам для разработки веб-приложений с использованием jQuery и к тому же является бесплатным. В следующих трех главах излагаются основы HTML, CSS и JavaScript. Если этот материал вам знаком, можете смело пропустить его и перейти непосредственно к главе 5, посвященной основным возможностям библиотеки jQuery.

---

[5] Если вместо заданного по умолчанию имени хоста `localhost` вы хотите использовать другое имя, обновите содержимое файла `hosts` в Windows (`C:\Windows\system32\drivers\etc\hosts`) или Linux (`/etc/hosts`), добавив в него запись, ссылающуюся на локальный хост (IP-адрес — 127.0.0.1), в данном случае — `127.0.0.1 node.jacquisflowershop.com`. Для внесения этих изменений необходимо обладать правами администратора. — *Примеч. ред.*

[6] При подготовке переводного издания книги во всех примерах для сервера Node.js использовался порт `9999`. — *Примеч. ред.*

ГЛАВА 2

Введение в HTML

На протяжении всей книги мы будем постоянно работать с HTML-документами. Материал данной главы необходим для понимания того, что мы будем делать в оставшейся части книги. Это не руководство по HTML, а скорее описание ключевых характеристик HTML, которые составят основу обсуждения в последующих главах.

Последняя версия HTML, известная под названием *HTML5*[1], уже сама по себе могла бы служить отдельной темой для изучения. Она содержит более ста элементов, каждый из которых имеет свое назначение и обладает собственной функциональностью. Для понимания принципов работы jQuery достаточно даже элементарных знаний HTML.

Базовый HTML-документ

Для начала давайте посмотрим, как выглядит HTML-документ. Это позволит вам получить первое представление о базовой структуре и иерархических принципах построения, которым подчиняется любой HTML-документ. Простой пример такого документа приведен в листинге 2.1. Этот документ используется в данной главе для того, чтобы познакомить вас с основными концепциями HTML.

Листинг 2.1. Пример простого HTML-документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <style>
        h1 {
            width: 700px; border: thick double black;
            margin-left: auto; margin-right: auto;
            text-align: center; font-size: x-large; padding: .5em;
            color: darkgreen; background-image: url("border.png");
            background-size: contain; margin-top: 0;
        }
        .dtable {display: table;}
        .drow {display: table-row;}
        .dcell {display: table-cell; padding: 10px;}
```

[1] Отсутствие пробела перед номером версии в названии спецификации, разрабатываемой группой WHATWG, не случайно. Подробный и увлекательный рассказ обо всех перипетиях развития HTML5 и о характере взаимного сотрудничества консорциума W3C и WHATWG можно найти по адресу `http://www.habrahabr.ru/blogs/webstandards/103256/`. — *Примеч. ред.*

```
        .dcell > * {vertical-align: middle}
        input {width: 2em; text-align: right;
               border: thin solid black; padding: 2px;}
        label {width: 6.5em; padding-left: .5em;
               display: inline-block;}
        #buttonDiv {text-align: center;}
        #oblock {display: block; margin-left: auto;
                 margin-right: auto; width: 700px;}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Несмотря на небольшой размер и простоту, этот документ позволяет выяснить ряд наиболее важных моментов, связанных с использованием HTML. Вид данного документа при его просмотре в окне браузера представлен на рис. 2.1[2].

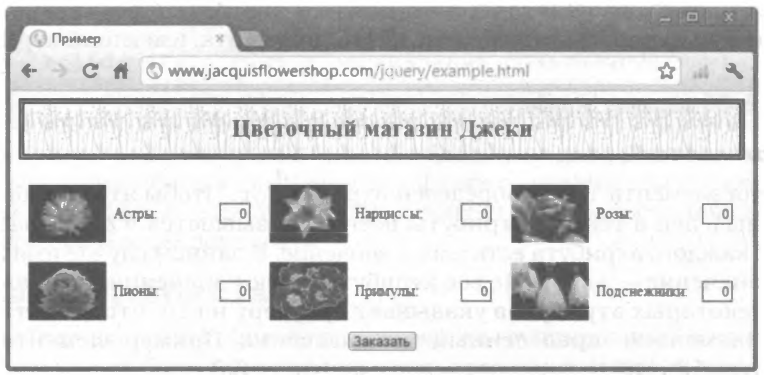


Рис. 2.1. Вид HTML-документа в окне браузера

Структура элементов HTML

В основе HTML лежит понятие *элемента*. Посредством элементов браузеру передается информация о типе содержимого каждой из частей HTML-документа. В качестве примера рассмотрим следующий элемент, взятый из приведенного выше примера.

```
<h1>Цветочный магазин Джеки</h1>
```

Этот элемент состоит из трех частей: открывающего (начального) дескриптора, закрывающего (конечного) дескриптора и содержимого (рис. 2.2).

Рис. 2.2. Структура простого HTML-элемента

Здесь `h1` — это *имя* данного элемента (также говорят *имя дескриптора*), которое указывает браузеру на то, что содержимое, заключенное между дескрипторами, должно интерпретироваться как заголовок верхнего уровня. Открывающий дескриптор создается путем заключения имени дескриптора в две угловые скобки (< и >). Точно так же создается и закрывающий дескриптор, только в этом случае после левой угловой скобки (<) дополнительно ставится косая черта (/).

---

[2] Для выполнения приведенных в книге примеров копируйте содержимое соответствующего листинга в файл `example.com`, который должен находиться в подкаталоге `jquery` корневого каталога вашего веб-сервера и в котором должны храниться также все необходимые вспомогательные файлы изображений. Кроме того, добавьте в файл `hosts` (в Windows — `C:\Windows\system32\drivers\etc\hosts`, в Linux — `/etc/hosts`) запись `127.0.0.1 www.jacquisflowershop.com` (требуются права администратора). — *Примеч. ред.*

Атрибуты

Браузеру можно предоставлять дополнительную информацию об элементах, снабжая их *атрибутами*. В качестве примера в листинге 2.2 показано, как выглядит один из элементов нашего образца HTML-документа, имеющий атрибут.

Листинг 2.2. Объявление атрибута

```
<label for="astor">Астры:</label>
```

Здесь для элемента `label` определен атрибут `for`. Чтобы атрибут был более заметен, он выделен в тексте. Атрибуты всегда указываются в открывающем дескрипторе. У каждого атрибута есть *имя* и *значение*. В данном случае имя атрибута — это `for`, а значение — `astor`. Не все атрибуты имеют значения; уже сам факт присутствия некоторых атрибутов указывает браузеру на то, что вы хотите связать с данным элементом определенный тип поведения. Пример элемента, которому присвоен атрибут такого типа, приведен в листинге 2.3.

Листинг 2.3. Объявление атрибута, не требующего указания значения

```
<input name="snowdrop" value="0" required>
```

В данном элементе определены три атрибута. Первым двум из них, `name` и `value`, присвоены значения, как это было сделано в предыдущем примере. (Имена этих атрибутов — `name` и `value` — не должны сбивать вас с толку: `snowdrop` — это значение атрибута `name`, а `0` — это значение атрибута `value`.) Третий атрибут представлен единственным словом — `required` ("требуется"). В данном случае мы имеем дело с атрибутом, задавать значение которого необязательно, хотя его и можно определить, указав значение, совпадающее с именем атрибута (`required="required"`), или использовав в качестве значения пустую строку (`required=""`).

Атрибуты `id` и `class`

В этой книге особое значение для нас будут иметь два атрибута: `id` и `class`. Одной из наиболее распространенных задач, с которыми приходится сталкиваться в процессе работы с jQuery, является определение местоположения одного или нескольких элементов в документе для последующего выполнения над ними некоторой операции. Атрибуты `id` (идентификатор) и `class` (класс) значительно облегчают поиск нужных элементов в документе.

Использование атрибута `id`

Атрибут `id` используется в качестве уникального идентификатора элемента в документе. В документе не должно быть двух элементов, имеющих одинаковые значения `id`. Пример простого документа, в котором используется атрибут `id`, представлен в листинге 2.4.

Листинг 2.4. Использование атрибута `id`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
```

```
</head>
<body>
    <h1 id="mainheader">Добро пожаловать в цветочный магазин
        Джеки!</h1>
    <h2 id="openinghours">Мы открыты с 10 до 18 часов ежедневно
        без выходных</h2>
    <h3 id="holidays">(закрыт в официальные праздничные дни)</h3>
</body>
</html>
```

В этом документе значения атрибута `id` определены для трех элементов. Значением атрибута `id` элемента `h1` является `mainheader`, элемента `h2` — `openinghours`, элемента `h3` — `holidays`. Атрибут `id` обеспечивает возможность поиска конкретных элементов в пределах документа.

Использование атрибута `class`

Атрибут `class` используется для произвольного группирования элементов. Один и тот же класс может быть присвоен многим элементам, а любой элемент может принадлежать одновременно нескольким классам, как показано в листинге 2.5.

Листинг 2.5. Использование атрибута `class`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
</head>
<body>
    <h1 id="mainheader" class="header">Добро пожаловать в
        цветочный магазин Джеки!</h1>
    <h2 class="header info">Мы открыты с 10 до 18 часов ежедневно
        без выходных</h2>
    <h3 class="info">(закрыт в официальные праздничные дни)</h3>
</body>
</html>
```

В этом примере элемент `h1` принадлежит классу `header`, элемент `h2` — классам `header` и `info`, а элемент `h3` — только классу `info`. Как видно из листинга, один и тот же элемент может входить сразу в несколько классов, имена которых указываются в виде списка с использованием пробела в качестве разделителя.

Содержимое элементов

Элементы могут включать не только текст, но и другие элементы. Ниже приведен пример элемента, внутри которого находятся другие элементы:

```
<div class="dcell">
    <img src="rose.png"/>
    <label for="rose">Розы:</label>
    <input name="rose" value="0" required>
</div>
```

Здесь в элемент `<div>` помещены три других элемента: `img`, `label` и `input`. Возможны несколько уровней *вложения* элементов, а не только один, как в данном примере. Вложение (подчинение) элементов — ключевая концепция HTML, поскольку

она подразумевает распространение функциональности внешнего элемента на элементы, содержащиеся внутри него (к обсуждению этой темы мы еще вернемся). Допускается смешивание текстового содержимого с другими элементами, как показано в следующем примере.

```
<div class="dcell">
    Это текстовое содержимое
    <img src="rose.png"/>
    Это дополнительный текст!
    <input name="rose" value="0" required>
</div>
```

Пустые элементы

Не все элементы могут иметь содержимое. Элементы, у которых не может быть содержимого, называются *пустыми* и записываются без отдельного закрывающего дескриптора. Вот как выглядит пустой элемент:

```
<img src="rose.png"/>
```

Пустой элемент определяется с помощью единственного дескриптора, в котором перед закрывающей угловой скобкой (>) помещается символ косой черты (/). Строго говоря, символ / должен отделяться от последнего символа последнего атрибута пробелом, как показано ниже.

```
<img src="rose.png" />
```

Однако в том, что касается интерпретации HTML-кода, браузеры весьма терпимы, так что символ пробела можно смело опускать. Пустые элементы часто используют, если элемент ссылается на внешний ресурс. В данном случае элемент `img` используется для ссылки на внешний файл изображения с именем `rose.png`.

Структура документа

Существуют ключевые элементы, которые определяют базовую структуру любого HTML-документа. Таковыми являются элементы `DOCTYPE`, `html`, `head` и `body`. В листинге 2.6 показано, как эти элементы соотносятся с остальной частью содержимого, которая для краткости опущена.

Листинг 2.6. Базовая структура HTML-документа

```
<!DOCTYPE html>
<html>
<head>
    ...содержимое элемента head...
</head>
<body>
    ...содержимое элемента body...
</body>
</html>
```

Каждый из этих элементов выполняет в HTML-документе свои специфические функции. Элемент `DOCTYPE` сообщает браузеру о том, что данный документ является HTML-документом, точнее — документом, соответствующим стандарту HTML5.

Более ранние версии HTML требуют указания дополнительной информации. Вот так, например, выглядит элемент DOCTYPE для документа HTML4.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```

Элемент html обозначает область документа, в которой находится HTML-содержимое. Этот элемент всегда содержит два других ключевых структурных элемента: head и body. Как подчеркивалось в начале главы, подробное изучение всех HTML-элементов не входит в наши планы. Число существующих элементов настолько велико, что для их полного описания в книге по HTML5 потребовалось более тысячи страниц. Учитывая это обстоятельство, я ограничусь лишь кратким описанием используемых элементов, чего, однако, будет вполне достаточно для того, чтобы вы хорошо понимали смысл документов, с которыми придется работать. В табл. 2.1 перечислены элементы, используемые в нашем образце документа (часть из них будет описана более подробно далее).

Таблица 2.1. HTML-элементы, входящие в образец документа

Элемент	*Описание*
DOCTYPE	Указывает тип содержимого текущего документа
body	Обозначает область документа, содержащую другие элементы (более подробно описывается далее)
button	Обозначает кнопку; часто используется для отправки на сервер элемента form
div	Универсальный элемент-контейнер; часто используется для дополнительного структурирования документа с целью улучшения его представления
form	Обозначает HTML-форму, позволяющую получать данные от пользователя и отправлять их на сервер для последующей обработки
h1	Обозначает заголовок
head	Обозначает область документа, содержащую метаданные (более подробно описывается далее)
html	Обозначает область документа, содержащую HTML-код (обычно это весь документ)
img	Обозначает изображение
input	Обозначает поле ввода, используемое для получения единичной порции данных от пользователя, и обычно является частью HTML-формы
script	Обозначает сценарий (обычно на языке JavaScript), который должен выполняться как часть документа
style	Обозначает область документа, содержащую параметры каскадных стилевых таблиц CSS (см. главу 3)
title	Обозначает название текущего документа; используется браузером для задания заголовка окна (или вкладки), в котором отображается содержимое документа

Элементы метаданных

Элемент head предназначен для хранения метаданных документа, иными словами — одного или нескольких элементов, которые описывают содержимое документа или воздействуют на него, но сами не отображаются браузером. В раздел head нашего образца документа включены три элемента метаданных: title, script и style. Самый простой из них — это элемент title. Его содержимое выводится браузером в качестве заголовка окна или вкладки, и его необходимо указывать

в любом HTML-документе. Два остальных элемента играют более важную роль в этой книге, и их смысл будет объяснен более подробно в последующих разделах.

Элемент `script`

С помощью элемента `script` можно включать в свой код сценарии JavaScript. Именно работе с этим элементом мы будем уделять больше всего внимания, как только перейдем к углубленному рассмотрению jQuery. В нашем образце документа имеется один элемент `script` (листинг 2.7).

Листинг 2.7. Элемент `script` из образца документа

```
<script src="jquery-1.7.js" type="text/javascript"></script>
```

Определяя для элемента `script` атрибут `src`, вы сообщаете браузеру, что хотите загрузить код JavaScript, содержащийся в другом файле. В данном случае этим файлом является основная библиотека jQuery, которую браузер загрузит из файла `jquery-1.7.js`. Один HTML-документ может содержать несколько элементов `script`, и при необходимости можно включить код сценария JavaScript непосредственно в документ, поместив его между открывающим и закрывающим дескрипторами элемента `script`, как показано в листинге 2.8.

Листинг 2.8. Включение встроенного кода JavaScript в документ с помощью элемента `script`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#mainheader').css("color", "red");
        });
    </script>
</head>
<body>
    <h1 id="mainheader" class="header">Добро пожаловать в
        цветочный магазин Джеки!</h1>
    <h2 class="header info">Мы открыты с 10 до 18 часов ежедневно
        без выходных</h2>
    <h3 class="info">(закрыт в официальные праздничные дни)</h3>
</body>
</html>
```

В этом примере присутствуют два элемента `script`. Первый из них импортирует в документ библиотеку jQuery, тогда как второй содержит простой сценарий, в котором используется базовая функциональность jQuery. Не старайтесь сейчас понять, как работает второй сценарий. В главе 5 мы приступим к систематическому изучению возможностей библиотеки jQuery. Элементы `script` могут включаться в элементы `head` и `body` HTML-документа. Как правило, в данной книге я буду помещать элементы `script` в элементы `head`, но это всего лишь дело вкуса.

Совет. Порядок следования элементов `script` имеет значение. Прежде чем использовать библиотеку jQuery, ее необходимо загрузить.

Элемент `style`

Элемент `style` обеспечивает один из нескольких возможных способов, с помощью которых в документ могут включаться свойства каскадных таблиц стилей (Cascading Style Sheets — CSS). Если говорить коротко, то свойства CSS позволяют управлять внешним видом документа при его отображении в браузере. Элемент `style`, входящий в состав образца документа, представлен вместе со своим содержимым в листинге 2.9.

Листинг 2.9. Использование элемента `style`

```
<style>
    h1 {
        width: 700px; border: thick double black;
        margin-left: auto; margin-right: auto;
        text-align: center; font-size: x-large; padding: .5em;
        color: darkgreen; background-image: url("border.png");
        background-size: contain; margin-top: 0;
    }
    .dtable {display: table;}
    .drow {display: table-row;}
    .dcell {display: table-cell; padding: 10px;}
    .dcell > * {vertical-align: middle}
    input {width: 2em; text-align: right;
           border: thin solid black; padding: 2px;}
    label {width: 6.5em;  padding-left: .5em;
           display: inline-block;}
    #buttonDiv {text-align: center;}
    #oblock {display: block; margin-left: auto;
             margin-right: auto; width: 700px;}
</style>
```

Браузер поддерживает набор свойств, значения которых используются для управления внешним видом каждого элемента. Элемент `style` позволяет выбирать элементы и изменять значения одного или нескольких таких свойств. Более подробно эта тема рассматривается в главе 3.

Как и элемент `script`, элемент `style` можно включать в элементы `head` и `body`, но в книге я всегда помещаю его в раздел `head`, как это сделано в образце документа. Такой подход также является делом вкуса; лично я предпочитаю, чтобы мои стили располагались в документе отдельно от его основного содержимого.

Элементы содержимого

В элемент `body` помещается *содержимое* HTML-документа. Под этим подразумеваются те элементы, которые браузер будет отображать для пользователя и на которые будут воздействуют такие элементы метаданных, как `script` и `style`.

Разделение семантики и представления

Одно из главных новшеств спецификации HTML5 носит философский характер: в этой спецификации большое значение придается отделению семантики элемента от его воздействия на способ представления содержимого. Это глубокая идея. Элементы HTML используются для структуризации содержимого и наделения его функциональностью, а также для последующего управления способом представления этого содержимого путем применения стилей CSS к элементам. Далеко не все

HTML-документы нуждаются в отображении (такие документы могут использоваться не только браузерами, но и автоматизированными программами), так что отделение представления HTML-документов от их содержимого упрощает их обработку и извлечение содержащейся в них информации автоматизированными методами.

Каждый HTML-элемент имеет определенный смысл. Например, элемент `article` используется для обозначения самостоятельной части содержимого, пригодной для синдикации, т.е. независимого распространения или многократного использования, тогда как элемент `h1` — для обозначения заголовка раздела содержимого.

Эта концепция составляет основу HTML. Элементы служат для обозначения типа соответствующего содержимого. Люди способны делать весьма точные заключения о смысле отдельных частей документа, исходя из контекста. Так, вы без труда определите, что заголовок некоторого раздела страницы подчинен предыдущему заголовку, если последний напечатан шрифтом большего размера. Компьютерам до этого пока еще далеко, и именно поэтому им облегчают задачу, помещая различные разделы содержимого в отдельные элементы для обозначения того, как они соотносятся между собой. Пример документа, в котором элементы используются для придания отдельным составляющим определенного структурного и смыслового значения, представлен в листинге 2.10.

Листинг 2.10. Использование HTML-элементов для придания структурного и смыслового значения содержимому

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
</head>
<body>
    <article>
        <header>
            <hgroup>
                <h1>Новая служба доставки</h1>
                <h2>Цвет и красота у ваших дверей</h2>
            </hgroup>
        </header>
        <section>
            Мы рады сообщить о начале предоставления новой
            услуги - доставки заказанных вами цветов прямо на дом.
            В радиусе 20 миль от магазина доставка осуществляется
            бесплатно, доплата за каждую дополнительную милю
            составляет $1. Мы гарантируем, что наши цветы вам
            понравятся. Дополнительную бесплатную консультацию
            вы можете получить по телефону.
        </section>
        <section>
            Новая услуга будет оказываться начиная
            со <b>среды</b>. Первым 50 покупателям предоставляется
            скидка $10.
        </section>
        <footer>
            <nav>
                Дополнительная информация:
                <a href="http://jacquisflowershop.com">
                    Узнайте больше о продукции</a>
            </nav>
```

```
        </footer>
    </article>
</body>
</html>
```

Каких-либо жестких правил, регламентирующих применение элементов `section` и `article`, не существует, однако весьма желательно, чтобы вы систематически применяли их в своих документах для семантической разметки содержимого. Эти элементы не предоставляют браузеру никакой информации относительно того, как должно отображаться содержимое, что отражает саму суть принципа разделения содержания и представления документа. В отношении большинства HTML-элементов браузеры руководствуются *соглашениями о стилях*, определяющими, как именно должны отображаться элементы, если их представление не было изменено стилями CSS, однако подразумевается, что именно широкое использование стилей CSS будет обеспечивать внешний вид документа. Это можно сделать как с помощью элемента `style`, так и средствами библиотеки jQuery, которая обеспечивает простой способ решения данной задачи с помощью элемента `script`.

Некоторые из элементов, определенных в спецификации HTML 4, создавались тогда, когда идея о необходимости разделения содержимого и представления документа еще не созрела, в результате чего иногда возникают двусмысленные ситуации. В качестве примера можно привести элемент `b`. До появления спецификации HTML5 элемент `b` сообщал браузеру о том, что содержимое, заключенное между его начальным и конечным дескрипторами, следует отображать полужирным начертанием. В спецификации HTML5, которая не поощряет использования элементов, управляющих исключительно внешним видом содержимого, дается новое определение этого элемента. Вот оно.

> Элемент `b` представляет фрагмент текста, некоторым образом выделенный относительно окружающего его содержимого, однако, в отличие, например, от выделения ключевых слов в рефератах документов или названий продуктов в обзорах, такое выделение, общепринятым типографским способом реализации которого является использование для текста полужирного начертания, не предполагает придания данному тексту какого-либо дополнительного логического акцента или подчеркивания степени его важности.
>
> — HTML: The Markup Language, w3c.org

С помощью этой довольно витиеватой формулировки нам просто пытаются сказать, что элемент `b` инструктирует браузер о том, что данный текст следует выделить полужирным начертанием. Элемент `b` не несет никакой семантической нагрузки, и речь здесь идет исключительно о способе представления текста. Туманность этого определения позволяет сделать один важный попутный вывод относительно стандарта HTML5: он все еще находится в стадии становления. Мы хотели бы, чтобы принцип отделения элементов от их представления соблюдался в полной мере, но реальность такова, что одновременно требуется обеспечить совместимость стандарта с бесчисленным количеством уже имеющихся документов, написанных с использованием более ранних версий HTML, и поэтому мы вынуждены идти на определенный компромисс.

Элементы `form` и `input`

Одним из наиболее интересных элементов в теле образца документа является элемент `form`. Этот элемент представляет собой механизм, с помощью которого можно получать от пользователей данные для их отправки на сервер. Как вы уви-

дите в главе 13, библиотека jQuery предоставляет великолепную поддержку для работы с формами, что обеспечивается как средствами ядра библиотеки, так и некоторыми широко используемыми плагинами. Элемент body, входящий в состав образца документа, представлен вместе со своим содержимым в листинге 2.11, в котором элемент form выделен полужирным шрифтом.

Листинг 2.11. Содержимое образца документа

```
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        <script type="text/javascript">
        $(document).ready(function() {
            var srcValue = $('img').attr('src');
            console.log("Значение атрибута: " + srcValue);
        });
```

```
    </script> </div>
    </form>
</body>
```

Везде, где встречается элемент `form`, обычно присутствует и элемент `input`. Он используется для получения определенной порции данных от пользователя. Элемент `input`, входящий в образец документа, представлен в листинге 2.12.

Листинг 2.12. Использование элемента `input`

```
<input name="snowdrop" value="0" required>
```

В данном случае элемент `input` получает от пользователя значение элемента данных `snowdrop`, инициализированного нулевым значением. Атрибут `required` сообщает браузеру о том, что пользователь не должен иметь возможности отправить форму на сервер, не предоставив предварительно значение элемента данных. Эта новая возможность, которая носит название *валидации* (проверки допустимости) формы, была впервые предусмотрена в HTML5, но, по правде говоря, валидация данных, обеспечиваемая jQuery, гораздо более эффективна, как это будет продемонстрировано в главе 13.

С формами тесно связан элемент `button`, который часто используется для отправки формы на сервер (а также может быть использован для сброса формы в исходное состояние). Фрагмент примера документа, в котором определен элемент `button`, приведен в листинге 2.13.

Листинг 2.13. Использование элемента `button`

```
<button type="submit">Заказать</button>
```

Указав для атрибута `type` значение `submit`, мы сообщаем браузеру о том, что нажатие кнопки должно приводить к отправке формы на сервер. Содержимое элемента `button` отображается в браузере поверх соответствующего элемента управления, как показано на рис. 2.3.

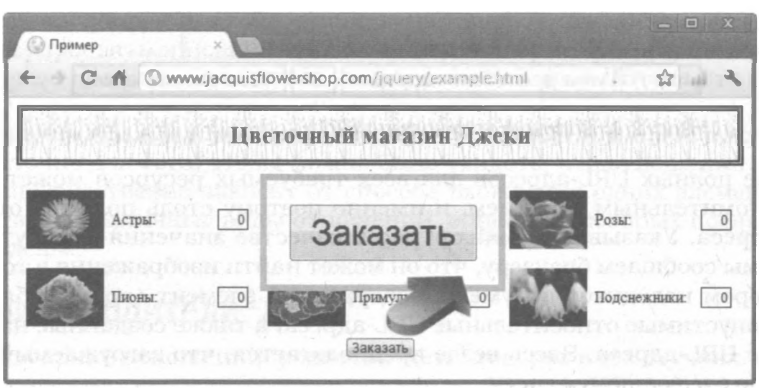


Рис. 2.3. Использование содержимого элемента `button`

Структурные элементы

Вы должны были заметить, что в теле образца документа встречается множество элементов `div`. Этот элемент не несет в себе определенного семантического

смысла и часто используется для управления компоновкой содержимого вебстраницы. В данном случае элемент `div` используется для *табличной компоновки* содержимого, так что элементы, содержащиеся внутри элементов `div`, отображаются для пользователя в виде макетной сетки. Компоновка страницы осуществляется за счет применения к элементам `div` стилей CSS, содержащихся в элементе `style`. Краткие вводные сведения о каскадных таблицах стилей CSS, интенсивно используемых в книге, содержатся в главе 3.

Элементы для работы с внешними ресурсами

Существует ряд элементов, позволяющих включать в документы внешние ресурсы. Хорошим примером может служить элемент `img`, который используют для добавления изображений в документ. В нашем документе он используется для включения в состав содержимого изображений цветов, предлагаемых к продаже, как показано в листинге 2.14.

Листинг 2.14. Использование элемента `img` для добавления ссылки на изображение, хранящееся во внешнем файле

```
<img src="snowdrop.png"/>
```

Для указания изображения используется атрибут `src`. В данном случае нужное изображение содержится в файле `snowdrop.png`. Мы используем здесь *относительный* URL-адрес, т.е. адрес, определенный относительно URL документа, содержащего данный элемент.

Альтернативой относительным URL-адресам являются *абсолютные* URL-адреса (называемые также *полностью определенными*, *уточненными* или *полными* URL-адресами). Этот термин относится к URL-адресам, в которых указаны все базовые компоненты, как показано на рис. 2.4. (На рисунке указан также порт, но если он опущен, то браузер будет использовать порт, заданный по умолчанию для данной схемы. Для схемы `http` таковым является порт 80.)

Рис. 2.4. Базовая структура URL

Указание полных URL-адресов для всех требуемых ресурсов может оказаться довольно утомительным занятием, и именно поэтому столь полезны относительные URL-адреса. Указывая `snowdrop.png` в качестве значения атрибута `src` элемента `img`, мы сообщаем браузеру, что он может найти изображения в том же каталоге, в котором находится документ, содержащий элемент `img`. В табл. 2.2 представлены допустимые относительные URL-адреса, а также созданные на их основе абсолютные URL-адреса. Здесь везде предполагается, что загружаемый документ находится по следующему адресу:

```
http://www.jacquisflowershop.com/jquery/example.html
```

Последний из приведенных в таблице примеров редко используется на практике, поскольку обеспечиваемая им экономия времени при ручном вводе информации незначительна, но он может оказаться полезным, если вы хотите быть уверены в том, что при запросе ресурсов используется та же схема, что и при извлечении

Таблица 2.2. Форматы относительных URL-адресов

Относительный URL-адрес	*Полный адрес*
`snowdrop.png`	`http://www.jacquisflowershop.com/jquery/snowdrop.png`
`/snowdrop.png`	`http://www.jacquisflowershop.com/snowdrop.png`
`/`	`http://www.jacquisflowershop.com/`
`//www.mydomain.com/index.html`	`http://www.mydomain.com/index.htm`

основного документа. Это позволяет избежать проблем в тех случаях, когда одна часть содержимого запрашивается по зашифрованному соединению (с использованием схемы `https`), а другая — по незашифрованному (с использованием схемы `http`). Некоторые браузеры, особенно Internet Explorer, плохо воспринимают смешивание безопасного и небезопасного содержимого, и в тех случаях, когда это происходит, выводят для пользователя соответствующее сообщение.

Предупреждение. Для навигации относительно того каталога, в котором хранится основной HTML-документ на веб-сервере, допустимо использование двух следующих подряд символов точки (`..`). Я не рекомендую применять этот прием хотя бы по той причине, что из соображений безопасности запросы, содержащие эти символы, будут отвергаться многими веб-серверами.

Иерархия элементов

В HTML-документе элементы естественным образом образуют иерархическую структуру. Элемент `html` содержит элемент `body`, который включает в себя элементы содержимого, каждый из которых содержит другие элементы и т.д.

Знание этой иерархии приобретает особое значение, если вы пытаетесь осуществлять навигацию по документу с помощью стилей CSS (о чем говорится в главе 3) или использовать средства библиотеки jQuery для поиска элементов в документе (что подробно обсуждается в главах 5 и 6).

Наиболее важную часть этой иерархии составляют отношения между элементами. Чтобы упростить описание этих отношений, на рис. 2.5 представлено иерархическое дерево для ряда элементов, содержащихся в документе сайта цветочного магазина.

На рисунке изображена лишь часть иерархии элементов нашего документа, которой достаточно для того, чтобы можно было увидеть, что отношения между элементами непосредственно зависят от способа вхождения одних элементов в другие. Существуют различные виды отношений, описанию которых посвящены следующие разделы.

Отношения "родители–дети"

О существовании отношений "родители–дети" говорят в тех случаях, когда один элемент содержится непосредственно в другом. На приведенном выше рисунке элемент `form` является *дочерним* (child) по отношению к элементу `body`. Это утверждение можно обратить: элемент `body` является *родительским* (parent) по отношению к элементу `form`. У одного элемента может быть несколько дочерних элементов, но родительский элемент может быть у каждого элемента только один. В рассматриваемом примере элемент `body` имеет два дочерних элемента (`form` и `h1`), и он является родительским по отношению к каждому из них.

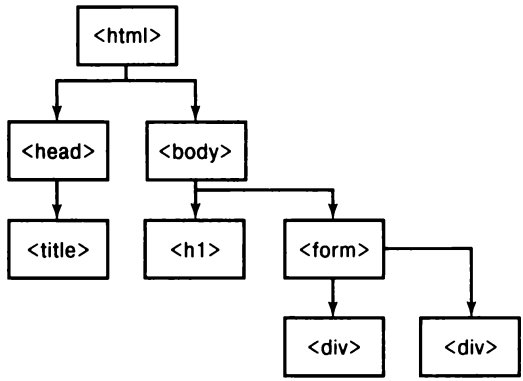


Рис. 2.5. Часть иерархического дерева документа

Отношения "родители–дети" существуют только между элементами и теми элементами, которые содержатся непосредственно внутри них. Так, например, элементы `div` являются дочерними элементами `form`, но не `body`.

Различают некоторые разновидности дочерних отношений. *Первый дочерний элемент* — это дочерний элемент, который первым встречается в документе. Например, элемент `h1` является первым дочерним элементом для элемента `body`. Соответственно, *последний дочерний элемент* — это дочерний элемент, который встречается в документе последним. Так, элемент `form` является последним дочерним элементом для элемента `body`. Также можно говорить об *n-м дочернем элементе* (первому дочернему элементу соответствует n = 1).

Отношения "предки–потомки"

К *потомкам* элемента относятся его дочерние элементы, дочерние элементы его дочерних элементов и т.д. Фактически любой элемент, непосредственно или косвенно содержащийся в некотором элементе, является потомком последнего. Например, потомками элемента `body` являются элементы `h1`, `form` и оба элемента `div`, а все изображенные на рисунке элементы являются потомками элемента `html`.

Антиподами потомков являются *предки* элемента, к которым относятся его родительский элемент, родительский элемент его родительского элемента и т.д. Например, предками элемента `form` являются элементы `body` и `html`. Оба элемента `div` имеют один и тот же набор предков: `form`, `body` и `html`.

"Сестринские" отношения

Сестринскими (sibling) называют отношения, которые существуют между элементами, имеющими общего родителя. В образце документа элементы `h1` и `form` — сестринские, поскольку элемент `body` является родительским по отношению к ним обоим. Работая с сестринскими элементами, мы будем употреблять выражения *предыдущий сестринский элемент* и *следующий сестринский элемент*. Таковыми являются сестринские элементы, которые встречаются в документе соответственно до или после текущего элемента. Не у всех элементов имеются как предыдущий, так и следующий сестринские элементы. Первый и последний дочерние элементы могут иметь соседний элемент лишь одного из этих типов.

Объектная модель документа

В процессе загрузки и обработки HTML-документа браузер создает *объектную модель документа* (Document Object Model — DOM). DOM — это модель, в которой каждый элемент документа представляется объектом JavaScript, но одновременно это и механизм, обеспечивающий возможность взаимодействия с содержимым HTML-документа программными средствами.

Примечание. В принципе, DOM может использоваться с любым языком программирования, реализованным в браузере. В основных типах браузеров преобладает язык JavaScript, и поэтому я не провожу различий между DOM как абстрактной идеей и DOM как коллекцией соответствующих объектов JavaScript.

Одной из причин, по которым вы должны хорошо представлять себе отношения между элементами, описанные в предыдущем разделе, является то, что те же соотношения сохраняются и в DOM. Отсюда следует, что природа и структура документа могут быть исследованы путем обхода узлов DOM-дерева с помощью JavaScript.

Совет. Использование DOM равносильно использованию JavaScript. Если вам требуется освежить свои знания по JavaScript, обратитесь к главе 4.

Далее демонстрируются некоторые базовые возможности DOM. В остальной части книги для доступа к объектам DOM будет использоваться jQuery, но в данном разделе я познакомлю вас с некоторыми возможностями встроенной поддержки и, в частности, продемонстрирую, насколько более элегантен подход, предлагаемый jQuery.

Использование DOM

В JavaScript объектом, который определяет базовую функциональность, доступную в DOM для всех типов элементов, является объект `HTMLElement`. Объект `HTMLElement` определяет свойства и методы, являющиеся общими для всех типов HTML-элементов, включая свойства, перечисленные в табл. 2.3.

Таблица 2.3. Базовые свойства объекта `HTMLElement`

Свойство	*Описание*	*Тип возвращаемого значения*
`className`	Возвращает или задает список классов, которым принадлежит данный элемент	`string`
`id`	Возвращает или задает значение атрибута `id`	`string`
`lang`	Возвращает или задает значение атрибута `lang`	`string`
`tagName`	Возвращает имя дескриптора (указывающее тип элемента)	`string`

Общее число доступных свойств гораздо больше. Их точное количество зависит от используемой версии HTML. Но указанных четырех свойств вполне достаточно для того, чтобы продемонстрировать базовые возможности DOM.

Для представления уникальных характеристик элементов каждого типа DOM использует объекты, получаемые путем наследования объекта `HTMLElement`. На-

пример, объект HTMLImageElement, используемый для представления в DOM элементов img, определяет свойство src, которому соответствует атрибут src элемента img. Я не буду углубляться в детальное описание всех объектов, соответствующих конкретным элементам, а лишь замечу, что, как правило, можно рассчитывать на то, что для интересующего вас атрибута всегда найдется соответствующее свойство объекта DOM.

Доступ к DOM осуществляется через глобальную переменную document, возвращающую объект Document. Объект Document представляет HTML-документ, который отображается в браузере, и определяет некоторые методы, обеспечивающие поиск объектов в DOM (табл. 2.4).

Таблица 2.4. Методы объекта Document, предназначенные для поиска элементов

Метод	*Описание*	*Тип возвращаемого значения*
getElementById(*<id>*)	Возвращает элемент с указанным значением id	HTMLElement
getElementsByClassName(*<класс>*)	Возвращает элементы с указанным значением класса	HTMLElement[]
getElementByTagName(*<дескриптор>*)	Возвращает элементы указанного типа	HTMLElement[]
querySelector(*<селектор>*)	Возвращает первый найденный элемент, соответствующий указанному селектору CSS	HTMLElement
querySelectorAll(*<селектор>*)	Возвращает все элементы, соответствующие указанному селектору CSS	HTMLElement[]

Вновь повторюсь, что здесь приведены лишь методы, которые используются в данной книге. В двух последних методах, представленных в таблице, используются селекторы CSS, описанные в главе 3. Пример использования объекта Document для поиска элементов определенного типа в документе приведен в листинге 2.15.

Листинг 2.15. Поиск элементов в DOM

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <style>
        h1 {
            width: 700px; border: thick double black;
            margin-left: auto; margin-right: auto;
            text-align: center; font-size: x-large; padding: .5em;
            color: darkgreen; background-image: url("border.png");
            background-size: contain; margin-top: 0;
        }
        .dtable {display: table;}
        .drow {display: table-row;}
        .dcell {display: table-cell; padding: 10px;}
        .dcell > * {vertical-align: middle}
```

```
        input {width: 2em; text-align: right;
               border: thin solid black; padding: 2px;}
        label {width: 6.5em;  padding-left: .5em;
               display: inline-block;}
        #buttonDiv {text-align: center;}
        #oblock {display: block; margin-left: auto;
                 margin-right: auto; width: 700px;}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
    <script>
        var elements = document.getElementsByTagName("img");
        for (var i = 0; i < elements.length; i++) {
```

```
            console.log("Элемент: " + elements[i].tagName +
                " " + elements[i].src);
        }
    </script>
</body>
</html>
```

В этом примере элемент `script` добавляется в конце элемента `body`. Когда браузер встречает элемент `script`, он сразу же приступает к выполнению операторов JavaScript, не дожидаясь окончания загрузки и обработки остальной части документа. Это становится проблемой, когда вы работаете с DOM, потому что вы пытаетесь выполнить поиск элементов посредством объекта `Document` еще до того, как в модели созданы интересующие вас объекты. Чтобы избежать этого, я поместил элемент `script` в конце документа. Библиотека jQuery предлагает элегантный способ решения этой проблемы, описанный в главе 5.

Для поиска в документе всех элементов `img` в сценарии используется метод `getElementsByTagName()`. Этот метод возвращает массив объектов, которые перебираются в цикле для вывода на консоль значений свойств `tagName` и `src` каждого объекта. Выводимая на консоль информация имеет следующий вид.

```
Элемент: IMG http://www.jacquisflowershop.com/jquery/astor.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/rose.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/peony.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/primula.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png
```

Изменение DOM

Объекты DOM *активны* в том смысле, что изменение значения свойства объекта оказывает влияние на документ, отображаемый в браузере. Сценарий, позволяющий продемонстрировать этот эффект, приведен в листинге 2.16. (Чтобы избежать ненужного повторения большей части документа, в листинге представлен лишь элемент `script`. Остальная часть документа остается той же, что и в предыдущем примере.)

Листинг 2.16. Изменение свойства DOM-объекта

```
...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        elements[i].src = "snowdrop.png";
    }
</script>
...
```

В этом сценарии значение атрибута `src` для всех элементов `img` устанавливается равным `snowdrop.png`. Результат выполнения сценария показан на рис. 2.6.

Рис. 2.6. Использование DOM для изменения HTML-документа

Изменение стилей

DOM можно использовать для изменения значений свойств CSS. (При необходимости можете обратиться к главе 3, в которой содержится краткое введение в CSS.) В программном интерфейсе DOM предусмотрена основательная поддержка CSS, но проще всего это можно сделать, используя свойство `style` объекта `HTMLElement`. Свойства объекта, возвращаемого свойством `style`, соответствуют свойствам CSS (я осознаю, что слово "свойство" встречается в этом коротком предложении слишком уж часто, за что приношу свои извинения).

Принятые в CSS правила присвоения имен свойствам несколько отличаются от тех, которые используются в объекте, возвращаемом свойством `style`. Например, свойству CSS `background-color` соответствует свойство `style.backgroundColor` этого объекта. Управление стилями документа с помощью DOM продемонстрировано в листинге 2.17.

Листинг 2.17. Использование DOM для изменения стилей элементов

```
...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        if (i > 0) {
            elements[i].style.opacity = 0.5;
        }
    }
</script>
...
```

Этот сценарий изменяет прозрачность, точнее — значение параметра непрозрачности (`opacity`), всех элементов `img` в документе, кроме первого. Один элемент `img` оставлен в неизменном виде, чтобы разница во внешнем виде элементов больше бросалась в глаза (рис. 2.7).

Обработка событий

События — это посылаемые браузером сигналы, уведомляющие об изменении состояния одного или нескольких объектов DOM. Для различных типов изменения

состояний предусмотрены различные события. Например, после щелчка на кнопке запускается событие `click`, а после отправки формы — событие `submit`. Многие события взаимосвязаны. Так, событие `mouseover` запускается при наведении указателя мыши на элемент, а событие `mouseout` — при выходе указателя мыши за пределы элемента.

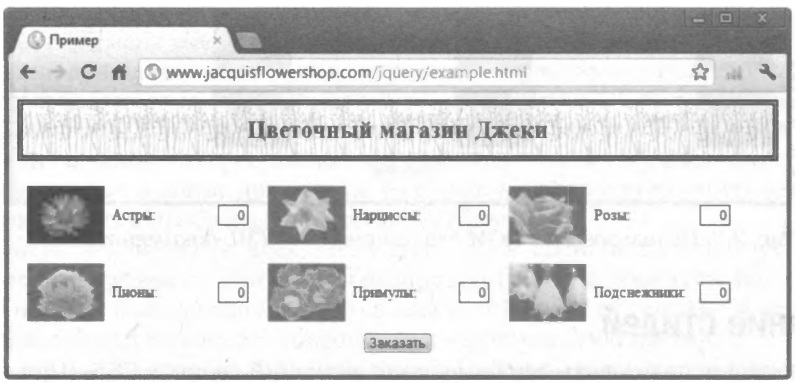


Рис. 2.7. Использование JavaScript для изменения значений свойств CSS

Можно отреагировать на события нужным образом, связав с событием для DOM-элемента некоторую функцию JavaScript. Соответствующий пример приведен в листинге 2.18.

Листинг 2.18. Обработка события

```
...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        elements[i].onmouseover = handleMouseOver;
        elements[i].onmouseout = handleMouseOut;
    }

    function handleMouseOver(e) {
        e.target.style.opacity = 0.5;
    }

    function handleMouseOut(e) {
        e.target.style.opacity = 1;
    }
</script>
...
```

В этом сценарии определены два обработчика событий, которые задаются значениями свойств `onmouseover` и `onmouseout` DOM-объектов `img`. В результате работы сценария изображения, представленные в документе, будут становится частично прозрачными при наведении на них указателя мыши и восстанавливать свой обычный вид при выводе указателя мыши за пределы области изображения. Пока что мы не будем обсуждать механизм обработки событий, используемый в DOM, поскольку подробному рассмотрению поддержки событий в jQuery посвящена глава 9. Тем не менее вам будет полезно уже сейчас получить предваритель-

ное представление об объекте Event, который передается обработчикам событий. Наиболее важные свойства этого объекта приведены в табл. 2.5.

Таблица 2.5. Функции и свойства объекта Event

Имя	*Описание*	*Тип возвращаемого значения*
type	Имя события, например mouseover	string
target	Целевой элемент события	HTMLElement
currentTarget	Элемент, приемники событий которого в данный момент активизируются	HTMLElement
eventPhase	Текущая фаза жизненного цикла события	number
bubbles	Возвращает значение true, если событие является всплывающим; в противном случае — значение false	boolean
cancelable	Возвращает значение true, если для события предусмотрено действие по умолчанию, которое можно отменить; в противном случае — значение false	boolean
stopPropagation()	Предотвращает распространение события по дереву элементов после запуска приемников событий для текущего элемента	void
stopImmediatePropagation()	Немедленно прекращает распространение события вверх или вниз по дереву элементов. Незапущенные приемники событий для текущего элемента игнорируются	void
preventDefault()	Предотвращает выполнение браузером действия по умолчанию, связанного с событием	void
defaultPrevented	Возвращает true, если перед этим вызывалась функция preventDefault()	boolean

В последнем примере элемент, к которому относится событие, определяется с помощью свойства target. Некоторые другие члены объекта Event связаны с *потоком события* и *действиями по умолчанию*, о чем рассказывается (очень кратко) в двух следующих разделах. В данной главе лишь закладывается фундамент для более глубокого обсуждения этой темы в последующих главах.

Поток события

Жизненный цикл события включает три фазы: *захват события* (capture), *передача события цели* (target) и *всплытие события* (bubbling). Когда запускается событие, браузер определяет элемент, к которому оно относится, — так называемый *целевой элемент* (цель) события. Далее браузер определяет все элементы, располагающиеся в иерархической структуре DOM-дерева между элементом body и целевым элементом события, и выясняет в отношении каждого из них, имеют ли они обработчики событий, требующие уведомления их о событиях, относящихся к подчиненным элементам (потомкам). Любой такой обработчик будет запускаться браузером до запуска обработчиков собственно целевого объекта события. (О том,

как выполнить запрос уведомления о событиях подчиненных элементов, говорится в главе 9.)

За фазой захвата следует фаза передачи события целевому элементу — простейшая из трех фаз. После завершения фазы захвата браузер запускает приемники событий данного типа, которые были добавлены в целевой элемент.

По завершении фазы передачи события целевому элементу браузер начинает последовательно просматривать всю цепочку элементов более высокого уровня (предков), продвигаясь в направлении элемента `body`. В ходе этого процесса браузер проверяет, существуют ли для каждого из просматриваемых элементов приемники событий данного типа, для которых захват событий отключен (подробнее об этом говорится в главе 9). Всплытие поддерживается не всеми событиями. Используя свойство `bubbles`, можно проверить, будет ли событие всплывать. Если значение этого свойства равно `true`, событие будет всплывать, в противном случае — не будет.

Действия браузера по умолчанию

Некоторые события определяют действие по умолчанию, которое будет выполняться в случае запуска события. Например, действием по умолчанию для события `click` является загрузка содержимого, местонахождение которого задано URL-адресом в атрибуте `href` элемента, на котором выполнен щелчок. Если для события предусмотрено действие по умолчанию, то значением свойства `cancelable` этого события будет `true`. Выполнение действия по умолчанию можно отменить, вызвав метод `preventDefault()`. Заметьте, что вызов метод `preventDefault()` не приводит к прекращению процесса прохождения события через фазы захвата события, его передачи целевому объекту и всплытия. Эти фазы по-прежнему будут выполняться, но по окончании фазы всплытия браузер не выполнит действие, предусмотренное по умолчанию. Можно определить, была ли вызвана для события функция `preventDefault()` одним из предшествующих обработчиков событий, получив значение свойства `defaultPrevented`. Если данное значение окажется равным `true`, то это будет означать, что функция `preventDefault()` вызывалась.

Резюме

В этой главе был приведен лишь краткий обзор принципов работы HTML, поскольку подробное описание каждого из более чем 100 элементов, которые могут входить в состав HTML-документа, в рамках данной книги не представляется возможным. Было показано, как создается и структурируется простой HTML-документ, как в результате вложения, наряду с текстовым содержимым, одних элементов в другие естественным образом возникает иерархия элементов и как в этой иерархии устанавливаются отношения между элементами, в соответствии с которыми элементы могут выступать друг для друга в роли родительских, дочерних и сестринских элементов, а также элементов-предков и элементов-потомков.

ГЛАВА 3

Введение в CSS

С HTML тесно связаны каскадные таблицы стилей (Cascading Style Sheets — CSS), являющиеся именно тем средством, с помощью которого можно управлять внешним видом документа. Существуют две причины, по которым CSS отводится особая роль в jQuery. Первая из них — используя *селекторы CSS* (описаны далее), можно предоставлять jQuery инструкции, касающиеся поиска элементов в HTML-документе. Вторая причина — одной из самых распространенных задач, для выполнения которых привлекается jQuery, является изменение стилей CSS, примененных к элементам.

Существует более 130 *свойств CSS*, каждое из которых отвечает за определенный аспект представления элемента. Как и HTML-элементы, свойства CSS настолько многочисленны, что для их полного описания в данной книге просто не хватило бы места. Поэтому основное внимание будет уделено практическим аспектам работы со свойствами CSS и применению стилей к элементам.

Знакомство с CSS

В процессе отображения элементов на экране браузер определяет, как именно должен быть представлен тот или иной элемент, используя набор свойств, известных как *свойства CSS*. Обратимся к примеру простого документа, представленного в листинге 3.1.

Листинг 3.1. Простой HTML-документ

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
</head>
<body>
    <h1>Новая служба доставки</h1>
    <h2>Цвет и красота у ваших дверей</h2>
    <h2>(специальное предложение)</h2>
    <p>Мы рады сообщить о начале предоставления новой услуги -
    доставки заказанных вами цветов прямо на дом. В радиусе
    20 миль от магазина доставка осуществляется бесплатно, доплата
    за каждую дополнительную милю составляет $1.</p>
</body>
</html>
```

Вид этого документа в окне браузера представлен на рис. 3.1.

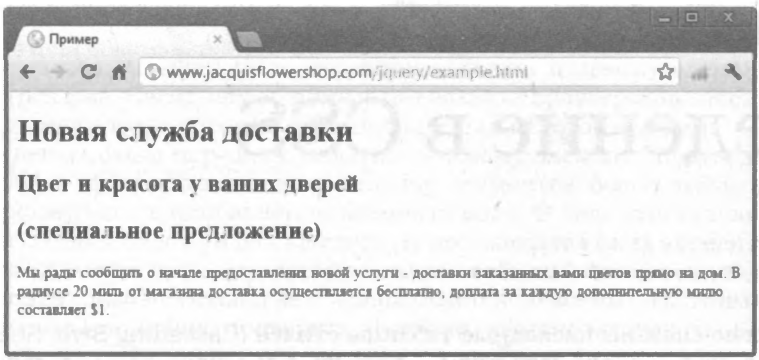


Рис. 3.1. Отображение простого документа в браузере

Свойства CSS слишком многочисленны для того, чтобы их можно было подробно рассмотреть по отдельности в данной книге. Однако многое о том, как они работают, можно узнать, рассмотрев лишь ограниченный набор свойств, описанных в табл. 3.1.

Таблица 3.1. Некоторые свойства CSS

Свойство	*Описание*
`color`	Устанавливает цвет переднего плана элемента (обычно определяет цвет текста)
`background-color`	Устанавливает цвет фона элемента
`font-size`	Устанавливает размер шрифта, используемого для вывода текста, содержащегося в элементе
`border`	Устанавливает цвет рамки (границ), окружающей элемент

Несмотря на отсутствие каких-либо значений, присвоенных свойствам, браузер смог отобразить содержимое, причем, как видно из рисунка, различные элементы содержимого представлены различным образом. Даже если значения свойств CSS браузеру не предоставляются, браузер самостоятельно применяет эти свойства, устанавливая их значения по умолчанию в соответствии с *соглашениями о стилях*, предусматривающими, как должен выглядеть тот или иной элемент, если его внешний вид не был явно определен. Это означает, что каждый браузер располагает набором значений свойств CSS, которые он применяет по умолчанию, если другие значения не предоставляются. Спецификация HTML определяет стили элементов для использования по умолчанию, однако разработчики браузеров не всегда твердо придерживаются положений спецификации, и поэтому одни и те же элементы в разных браузерах могут выглядеть по-разному. Значения свойств CSS, используемые по умолчанию в браузере Google Chrome, приведены в табл. 3.2.

Таблица 3.2. Некоторые свойства CSS и их значения, используемые по умолчанию браузером Google Chrome по отношению к различным элементам

Свойство	*h1*	*h2*	*p*
`color`	`black`	`black`	`black`
`background-color`	`transparent`	`transparent`	`transparent`
`font-size`	`2em`	`1.5em`	`16px`
`border`	`none`	`none`	`none`

Как нетрудно заметить, значения свойств `color`, `background-color` и `border` для всех трех элементов одинаковы; отличаются лишь значения свойства `font-size`. Далее будут описаны единицы измерения значений этих свойств и рассказано о том, почему свойство `font-size` выражается в единицах `em` в случае элементов `h1` и `h2` и в единицах `px` в случае элемента `p`. А пока что займемся установкой значений свойств, не заботясь о том, какие единицы используются для их измерения.

Встроенные стили

Самый непосредственный способ задания значений свойств CSS состоит в том, чтобы поместить атрибут `style` в элемент, представление которого требуется изменить. Такие стили называются *встроенными* (inline). Пример использования встроенного стиля приведен в листинге 3.2.

Листинг 3.2. Использование атрибута `style` для изменения значения свойства CSS элемента

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
</head>
<body>
    <h1>Новая служба доставки</h1>
    <h2 style="background-color: grey; color: white">Цвет и
        красота у ваших дверей</h2>
    <h2>(специальное предложение)</h2>
    <p>Мы рады сообщить о начале предоставления новой услуги -
        доставки заказанных вами цветов прямо на дом. В радиусе
        20 миль от магазина доставка осуществляется бесплатно,
        доплата за каждую дополнительную милю составляет $1.</p>
</body>
</html>
```

В этом примере для указания значений двух свойств CSS используются *объявления стилей*. Формат записи значения атрибута `style` показан на рис. 3.2.

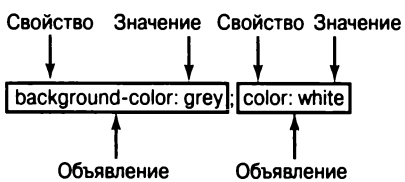


Рис. 3.2. Формат записи значения атрибута `style`

В каждом объявлении стиля указываются имя свойства и его значение, разделенные двоеточием (`:`). Можно записать несколько объявлений подряд, используя в качестве разделителя точку с запятой (`;`). В данном примере для свойства `background-color` устанавливается значение `grey`, а для свойства `color` — значение `white`. Эти значения задаются в атрибуте `style` элемента `h2` и будут воздействовать только на содержимое данного элемента (на другие элементы они никакого влияния оказывать не будут, даже если те также являются элементами `h2`). Резуль-

тат применения нового стиля с целью изменения представления первого элемента h2 показан на рис. 3.3.

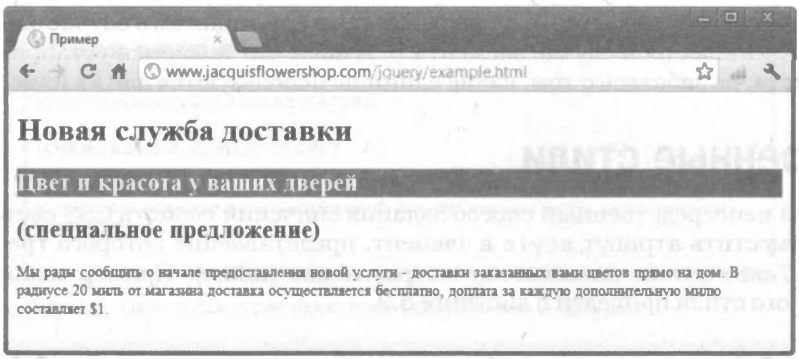


Рис. 3.3. Результат изменения значений свойств CSS в атрибуте `style` элемента `h2`

Внедренные стили

Использовать атрибут `style` несложно, однако он воздействует лишь на тот элемент, в котором определен. При желании можно систематически использовать этот подход в отношении всех элементов, представление которых требуется изменить, однако с увеличением количества изменяемых элементов возрастает и трудоемкость такого подхода, при этом вероятность появления ошибок очень быстро увеличивается, особенно если впоследствии документ будет многократно пересматриваться. Гораздо более эффективный метод состоит в том, чтобы определить *внедренный стиль* с помощью *элемента* (а не атрибута!) `style` и снабдить браузер инструкциями относительно того, к каким элементам данный стиль должен быть применен, используя *селекторы*. Пример определения внедренного стиля приведен в листинге 3.3.

Листинг 3.3. Определение внедренного стиля

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        h2 { background-color: grey; color: white;}
    </style>
</head>
<body>
    <h1>Новая служба доставки</h1>
    <h2>Цвет и красота у ваших дверей</h2>
    <h2>(специальное предложение)</h2>
    <p>Мы рады сообщить о начале предоставления новой услуги -
    доставки заказанных вами цветов прямо на дом. В радиусе
    20 миль от магазина доставка осуществляется бесплатно, доплата
    за каждую дополнительную милю составляет $1.</p>
</body>
</html>
```

Во внедренном стиле по-прежнему используются объявления стилей, однако в данном случае они заключаются в фигурные скобки ({ и }) и предваряются *селектором*. Формат записи внедренного стиля показан на рис. 3.4.

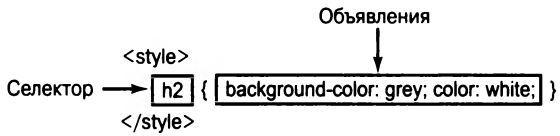


Рис. 3.4. Формат записи внедренного стиля

Совет. В данном случае элемент `style` располагается внутри элемента `head`, однако с равным успехом он мог бы находиться и внутри элемента `body`. Я отдаю предпочтение первому способу, так как являюсь приверженцем идеи отделения содержимого документа от метаданных.

Селекторы CSS играют очень важную роль в jQuery, поскольку они служат тем фундаментом, на основе которого вы выбираете элементы в DOM для выполнения операций над ними. В приведенном выше примере в качестве селектора используется элемент `h2`. Это означает, что объявления стилей, содержащиеся внутри скобок, будут применяться ко всем элементам `h2` в документе. Результат применения нового стиля показан на рис. 3.5.

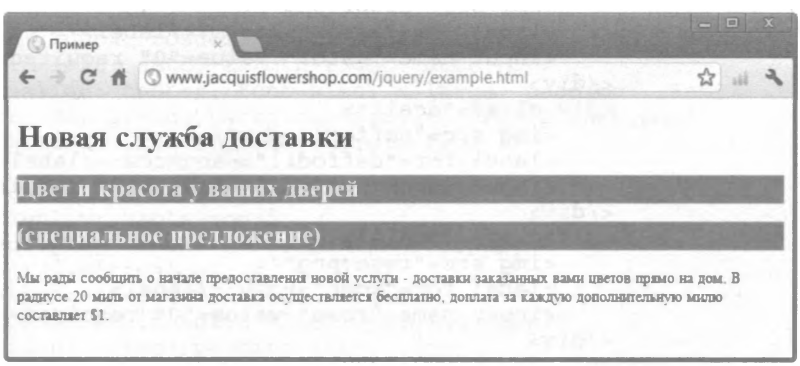


Рис. 3.5. Влияние внедренного стиля на внешний вид документа

Элемент `style` может содержать несколько внедренных стилей. Примером использования более сложного набора стилей может служить представленный в листинге 3.4 документ из главы 2, лежащий в основе сайта цветочного магазина.

Листинг 3.4. Пример использования в HTML-документе более сложного набора стилей

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <style>
        h1 {
            width: 700px; border: thick double black;
            margin-left: auto; margin-right: auto;
```

```
            text-align: center; font-size: x-large; padding: .5em;
            color: darkgreen; background-image: url("border.png");
            background-size: contain; margin-top: 0;
        }
        .dtable {display: table;}
        .drow {display: table-row;}
        .dcell {display: table-cell; padding: 10px;}
        .dcell > * {vertical-align: middle}
        input {width: 2em; text-align: right;
               border: thin solid black; padding: 2px;}
        label {width: 6.5em; padding-left: .5em;
               display: inline-block;}
        #buttonDiv {text-align: center;}
        #oblock {display: block; margin-left: auto;
                 margin-right: auto; width: 700px;}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
```

```
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

В данном примере элемент `style` содержит несколько внедренных стилей, причем некоторые из них, в частности тот, в котором используется селектор `h2`, определяют значения многих свойств.

Внешние таблицы стилей

Вместо того чтобы определять один и тот же набор стилей в каждом HTML-документе, в котором они должны применяться, можно использовать отдельную *таблицу стилей*. Под этим подразумевается независимый файл, обычно имеющий расширение `.css`, в который помещаются все необходимые стили. В листинге 3.5 представлено содержимое файла `style.css`, который объединяет все стили документа, приведенного в предыдущем примере.

Листинг 3.5. Содержимое файла `style.css`

```
h1 {
    width: 700px; border: thick double black;
    margin-left: auto; margin-right: auto;
    text-align: center; font-size: x-large; padding: .5em;
    color: darkgreen; background-image: url("border.png");
    background-size: contain; margin-top: 0;
}
.dtable {display: table;}
.drow {display: table-row;}
.dcell {display: table-cell; padding: 10px;}
.dcell > * {vertical-align: middle}
input {width: 2em; text-align: right;
       border: thin solid black; padding: 2px;}
label {width: 6.5em; padding-left: .5em;
       display: inline-block;}
#buttonDiv {text-align: center;}
#oblock {display: block; margin-left: auto;
         margin-right: auto; width: 700px;}
```

Использовать элемент `style` в таблице стилей необязательно. Можно просто определить в файле селектор и объявления. Привязка стилей к документу осуществляется с помощью элемента `link`, как показано в листинге 3.6.

Листинг 3.6. Подключение внешней таблицы стилей к документу

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
</head>
```

```
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

К документу может быть подключено несколько таблиц стилей, для каждой из которых используется свой элемент `link`. Последовательность, в которой таблицы стилей подключаются к документу, имеет значение, если для одного и того же селектора в них определены два разных стиля. Применяться будет стиль, загруженный последним.

Селекторы CSS

Наверное, вы обратили внимание на то, что селекторы в приведенном выше примере таблицы стилей имеют различную природу. Одни из них — это просто имена элементов (например, `h1` и `input`), тогда как другие начинаются с точки (например, `.dtable` и `.row`) или символа "решетки" (например, `#buttonDiv` и `#oblock`). Если же вы наблюдательны, то должны были заметить, что один из селекторов состоит из нескольких компонентов: `.dcell > *`. Каждый из селекторов CSS предназначен для выбора определенных элементов в документе, и то, каким образом браузер будет осуществлять поиск элементов, зависит от вида селектора. В этом разделе описаны виды селекторов, определенные в CSS. Мы начнем с рассмотрения базовых селекторов, описанных в табл. 3.3.

Таблица 3.3. Базовые селекторы

Селектор	*Описание*
*	Универсальный селектор, выбирает все элементы
<тип>	Выбирает элементы указанного типа
.<класс>	Выбирает элементы, имеющие указанный класс (независимо от типа элемента)
<тип>.<класс>	Выбирает элементы указанного типа, имеющие указанный класс
#<идентификатор>	Выбирает элементы, имеющие указанное значение атрибута `id`

Эти селекторы используются гораздо чаще, чем все остальные (например, ими охватываются почти все стили, определенные в нашем образце документа).

Селекторы атрибутов

Базовые селекторы позволяют работать с атрибутами `id` и `class` (описанными в главе 2), но существуют также селекторы, которые позволяют работать с любыми атрибутами. Их описание приведено в табл. 3.4.

Таблица 3.4. Селекторы атрибутов

Селектор	*Описание*
[атрибут]	Выбирает все элементы, для которых определен *атрибут*, независимо от его значения
[атрибут="значение"]	Выбирает все элементы, для которых определен *атрибут* с заданным *значением*
[атрибут^="значение"]	Выбирает все элементы, для которых определен *атрибут*, начинающийся со строки *значение*
[атрибут$="значение"]	Выбирает все элементы, для которых определен *атрибут*, заканчивающийся строкой *значение*
[атрибут="значение"]*	Выбирает все элементы, для которых определен *атрибут*, содержащий строку *значение*
[атрибут~="значение"]	Выбирает все элементы, для которых определен *атрибут*, в перечне значений которого присутствует строка *значение*
[атрибут|="значение"]	Выбирает все элементы, для которых определен *атрибут*, значением которого является список значений, разделенных символами дефиса, начинающийся строкой *значение*

Пример простого документа с внедренным стилем, в котором используются селекторы атрибутов, приведен в листинге 3.7.

Листинг 3.7. Использование селекторов атрибутов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        [lang] { background-color: grey; color: white;}
        [lang="es"] {font-size: 14px;}
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <h2 lang="es">(Color y belezza a tu puerta)</h2>
    <p>Мы рады сообщить о начале предоставления новой услуги -
    доставки заказанных вами цветов прямо на дом. В радиусе
    20 миль от магазина доставка осуществляется бесплатно, доплата
    за каждую дополнительную милю составляет $1.</p>
</body>
</html>
```

Первому селектору соответствует любой элемент, имеющий атрибут `lang`, а второму — любой элемент, значением атрибута `lang` которого является `es`. Результат применения этих стилей показан на рис. 3.6.

Рис. 3.6. Применение стилей, использующих селекторы атрибутов

Примечание. Относительно этого рисунка следует сделать одно важное замечание. Обратите внимание, что на один из элементов `h2` оказали воздействие *оба* встроенных стиля. Первый стиль применяется ко всем элементам, для которых определен атрибут `lang`. Второй стиль применяется ко всем элементам, для которых определен атрибут `lang`, имеющий значение `es`. Второй элемент `h2` в документе удовлетворяет обоим критериям, и поэтому изменяются значения всех трех его свойств: `background-color`, `color` и `font-size`. Более подробно об этом будет говориться при обсуждении каскадирования стилей.

Иерархические селекторы

Как уже отмечалось в главе 2, элементы (и представляющие их DOM-объекты) подчиняются определенной иерархии, которая порождает между ними отношения различного типа. Для выбора элементов на основании иерархических отношений между ними предусмотрены селекторы CSS, перечисленные в табл. 3.5.

Таблица 3.5. Иерархические селекторы

Селектор	*Описание*
<селектор> <селектор>	Выбирает элементы, которые соответствуют второму *селектору* и являются потомками элементов, соответствующих первому *селектору*
<селектор> > <селектор>	Выбирает элементы, которые соответствуют второму *селектору* и являются дочерними по отношению к элементам, соответствующим первому *селектору*
<селектор> + <селектор>	Выбирает элементы, которые соответствуют второму *селектору* и располагаются непосредственно за своим сестринским элементом, соответствующим первому *селектору*
<селектор> ~ <селектор>	Выбирает элементы, которые соответствуют второму *селектору* и при этом являются сестринскими по отношению к элементу, соответствующему первому *селектору* (и располагаются после него)

В рассматриваемом нами примере документа один из этих селекторов используется следующим образом:

```
.dcell > * {vertical-align: middle}
```

Этот селектор выбирает все элементы, которые являются дочерними по отношению к элементам, принадлежащим к классу `dcell`, а объявление стиля устанавливает для свойства `vertical-align` значение `middle`. Пример использования других иерархических селекторов приведен в листинге 3.8.

Листинг 3.8. Использование иерархических селекторов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        h1 ~ [lang] { background-color: grey; color: white;}
        h1 + [lang] {font-size: 12px;}
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <h2 lang="es">(Color y belezza a tu puerta)</h2>
    <p>Мы рады сообщить о начале предоставления новой услуги -
    доставки заказанных вами цветов прямо на дом. В радиусе
    20 миль от магазина доставка осуществляется бесплатно, доплата
    за каждую дополнительную милю составляет $1.</p>
</body>
</html>
```

В этом примере используются оба селектора сестринских элементов. Первому из них — тому, в котором используется символ тильды (~), — соответствует любой элемент, который имеет атрибут `lang`, является сестринским по отношению к элементу `h1` и определен после него. Применительно к нашему примеру документа это означает, что первый селектор выбирает оба элемента `h2` (потому что оба они удовлетворяют указанным условиям). Второй селектор — тот, в котором используется знак "плюс" (+), — аналогичен первому, однако выбирает лишь тот из элементов, являющихся сестринскими по отношению к элементу `h1`, который непосредственно следует за ним. Это означает, что выбирается лишь первый из двух элементов `h2`. Конечный результат представлен на рис. 3.7.

Рис. 3.7. Использование селекторов сестринских элементов

Селекторы псевдоклассов и псевдоэлементов

CSS поддерживает набор *селекторов псевдоэлементов* и *селекторов псевдоклассов*. Эти селекторы не предназначены для работы непосредственно с какими-либо реальными элементами или классами, существующими в документе, однако обеспечиваемая ими функциональность чрезвычайно удобна. Полный их перечень приведен в табл. 3.6.

Таблица 3.6. Псевдоселекторы

Селектор	Описание
`:active`	Выбирает элемент, в данный момент активизированный пользователем. Обычно таковым является элемент, на котором был выполнен щелчок мышью
`:checked`	Выбирает выделенные (выбранные пользователем) элементы
`:default`	Выбирает элементы, являющиеся выделенными по умолчанию
`:disabled`	Выбирает элементы, находящиеся в неактивном состоянии
`:empty`	Выбирает элементы, не содержащие дочерних элементов
`:enabled`	Выбирает элементы, находящиеся в активном состоянии
`:first-child`	Выбирает элементы, являющиеся первыми дочерними элементами своих родителей
`:first-letter`	Выбирает первую букву блока текста

Окончание табл. 3.6

Селектор	Описание
`:first-line`	Выбирает первую строку блока текста
`:focus`	Выбирает элемент, получивший фокус
`:hover`	Выбирает элемент, на который наведен указатель мыши
`:in-range` `:out-of-range`	Выбирают элементы `input` с ограничениями, если введенное значение поля соответственно находится в диапазоне допустимых значений или выходит за его пределы
`:lang(`*<язык>*`)`	Выбирает элементы по значению атрибута *язык*
`:last-child`	Выбирает элементы, являющиеся последними дочерними элементами своих родителей
`:link`	Выбирает элементы `link`
`:nth-child(`*n*`)`	Выбирает элементы, являющиеся *n*-ми дочерними элементами своих родителей
`:nth-last-child(`*n*`)`	Выбирает элементы, являющиеся *n*-ми, при отсчете с конца, дочерними элементами своих родителей
`:nth-last-of-type(`*n*`)`	Выбирает элементы, являющиеся *n*-ми, при отсчете с конца, дочерними элементами заданного типа для своих родителей
`:nth-of-type(`*n*`)`	Выбирает элементы, являющиеся *n*-ми дочерними элементами заданного типа для своих родителей
`:only-child`	Выбирает элементы, являющиеся единственными дочерними элементами своих родителей
`:only-of-type`	Выбирает элементы, являющиеся единственными дочерними элементами заданного типа для своих родителей
`:required` `:optional`	Выбирают элементы `input` соответственно по признаку обязательности или необязательности заполнения поля
`:root`	Выбирает корневой элемент документа
`:target`	Выбирает элемент, на который ссылается идентификатор фрагмента в URL-адресе
`:valid` `:invalid`	Выбирают элементы `input` на основе соответственно допустимости или недопустимости введенного в поле значения
`:visited`	Выбирает элементы `link`, соответствующие посещенным ссылкам

Пример использования некоторых из перечисленных селекторов приведен в листинге 3.9.

Листинг 3.9. Использование псевдоселекторов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        :nth-of-type(2) { background-color: grey; color: white;}
        p:first-letter {font-size: 40px;}
```

```
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <h2 lang="es">(Color y belezza a tu puerta)</h2>
    <p>Мы рады сообщить о начале предоставления новой услуги -
    доставки заказанных вами цветов прямо на дом. В радиусе
    20 миль от магазина доставка осуществляется бесплатно, доплата
    за каждую дополнительную милю составляет $1.</p>
</body>
</html>
```

Псевдоселекторы могут использоваться как сами по себе, так и в качестве модификаторов других селекторов. В этом примере представлены оба подхода. Первому селектору соответствует любой элемент, являющийся вторым дочерним элементом среди элементов своего типа. Первому селектору соответствует первая буква любого элемента p. Результат применения этих стилей можно увидеть на рис. 3.8.

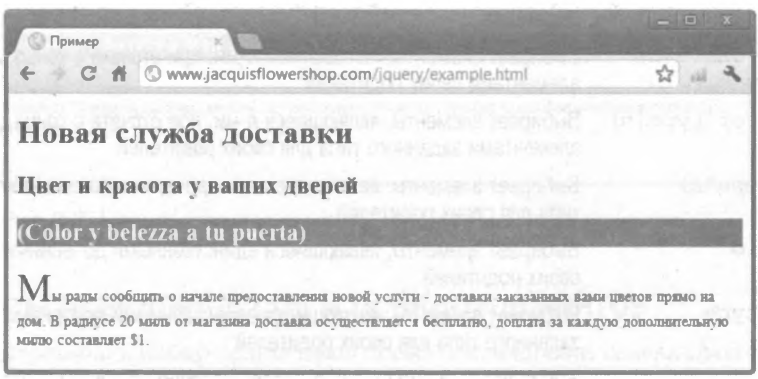


Рис. 3.8. Использование псевдоселекторов для применения стилей

Комбинированные и инверсные селекторы

Объединение селекторов обеспечивает значительно более гибкие возможности выбора элементов. В частности, можно объединять селекторы в группы или обращать (инвертировать) их действие на противоположное посредством их отрицания (инверсии). Оба этих подхода представлены в табл. 3.7.

Таблица 3.7. Гибкие комбинации селекторов

Селектор	Описание
`<селектор>, <селектор>`	Выбирает объединенную совокупность элементов, соответствующих первому селектору, и элементов, соответствующих второму селектору
`:not(<селектор>)`	Выбирает элементы, не соответствующие указанному селектору

Пример использования комбинированных и инверсных селекторов приведен в листинге 3.10.

Листинг 3.10. Использование комбинированных и инверсных селекторов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        h1, h2 { background-color: grey; color: white;}
        :not(html):not(body):not(:first-child)
            {border: medium double black;}
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <p>Мы рады сообщить о начале предоставления новой услуги -
    доставки заказанных вами цветов прямо на дом. В радиусе
    20 миль от магазина доставка осуществляется бесплатно, доплата
    за каждую дополнительную милю составляет $1.</p>
</body>
</html>
```

В данном примере первый селектор — это сгруппированные селекторы `h1` и `h2`. Как вы, вероятно, уже догадались, этот селектор выбирает все элементы `h1` и `h2` в документе. Второй селектор выглядит несколько необычно. На его примере я хотел продемонстрировать использование псевдоселекторов в качестве модификаторов других селекторов, в том числе и для их инверсии.

```
:not(html):not(body):not(:first-child)
    {border: medium double black;}
```

Этому селектору соответствует любой элемент, который не является элементом `html`, не является элементом `body` и не является первым дочерним элементом своего родителя. Результат применения указанных стилей представлен на рис. 3.9.

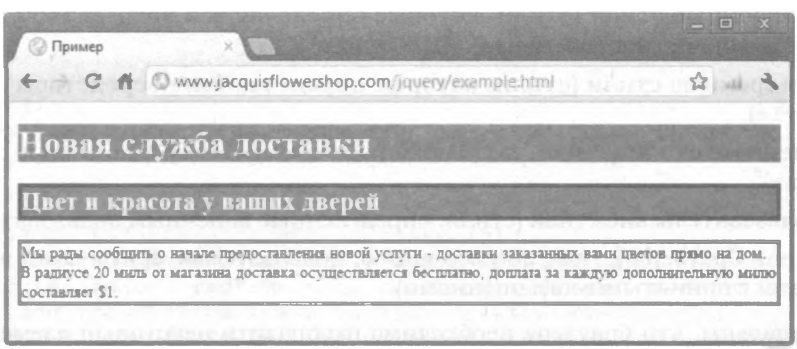


Рис. 3.9. Создание комбинированных и инверсных селекторов

Каскадирование стилей

Возможно существование нескольких источников CSS-свойств, каждый из которых связан с документом. Чтобы понять принципы, в соответствии с которыми браузер определяет, какие из этих значений следует использовать при отображе-

нии элементов, вам необходимо знать, что такое *каскадирование* и *наследование* стилей. Один из источников CSS-стилей — *авторские стили*, с тремя способами определения которых (непосредственно внутри элемента, встраивание в документ и подключение таблиц стилей из внешних файлов с помощью элемента `link`) вы уже познакомились[1], однако существуют также два дополнительных источника: стили браузера и пользовательские стили.

Стили браузера (правильнее — *стили пользовательского агента*) определяются соглашениями о стилях, которые браузер применяет к элементам, если никакие другие стили для них не были явно определены. Пример использования браузером соглашений о стилях уже приводился в начале главы.

Кроме того, большинство браузеров предоставляют конечным пользователям возможность определять собственные таблицы стилей. Такие стили называются *пользовательскими*. Данная возможность не относится к числу широко применяемых, но те пользователи, которые создают собственные таблицы стилей, обычно высоко ценят ее, и не в последнюю очередь потому, что такой подход позволяет сделать страницы более доступными.

Каждый браузер имеет собственный механизм для работы с пользовательскими стилями. Например, в версии Google Chrome для Windows в каталоге профиля пользователя создается файл `Default\User StyleSheets\Custom.css`. Любые стили, добавленные в этот файл, будут применяться к *любому* сайту, посещаемому пользователем, подчиняясь при этом правилам каскадирования, описанным в следующем разделе.

Что такое каскадирование стилей

Теперь, когда вам известны все источники стилей, которые должны учитываться браузером, можно поговорить о том, в какой очередности браузер просматривает источники стилей в поиске значения свойства, необходимого для отображения того или иного элемента. Браузер строго придерживается указанного ниже *каскадного* порядка просмотра, соответствующего порядку убывания приоритета стилей.

1. Встроенные стили (стили, которые определяются посредством атрибута `style` конкретного элемента).
2. Внедренные стили (стили, которые определяются посредством элемента `style`).
3. Внешние стили (стили, подключенные к документу с помощью элемента `link`)[2].
4. Пользовательские стили (стили, определенные конечным пользователем).
5. Стили браузера (стили по умолчанию, используемые браузерами в соответствии с принятыми соглашениями).

Предположим, что браузеру необходимо отобразить некоторый элемент `p`. Допустим далее, что браузер должен определить, какой цвет следует использовать для вывода текста. Для ответа на этот вопрос браузер должен найти значение CSS-свойства `color`. Прежде всего он должен проверить, определен ли для элемента,

---

[1] Подключение таблицы стилей, хранящейся во внешнем файле, возможно также с помощью директивы `@import`. Импортированные стили имеют более высокий приоритет по сравнению со стилями, подключенными с помощью дескриптора `<link>`. — *Примеч. ред.*

[2] См. предыдущую сноску. — *Примеч. ред.*

который он пытается отобразить, встроенный стиль, определяющий значение свойства color подобно тому, как показано ниже.

```
<p style="color: red">Мы рады сообщить о начале предоставления
    новой услуги - доставки заказанных вами цветов прямо на
    дом. В радиусе 20 миль от магазина доставка осуществляется
    бесплатно, доплата за каждую дополнительную милю составляет
    $1.</p>
```

В случае отсутствия нужного встроенного стиля браузер приступает к поиску элемента style, содержащего стиль, который определяет значение нужного свойства, например, следующим образом:

```
<style>
    p {color: red};
</style>
```

Если подходящий элемент style отсутствует, браузер просматривает таблицы стилей, загруженные с помощью элемента link, и так до тех пор, пока не найдет значение свойства color. Если же никаких иных значений найти не удается, будет использовано значение, определяемое стилями браузера по умолчанию.

Примечание. Первые три источника значений свойств (встроенные стили, внедренные стили и внешние таблицы стилей) совокупно называются *авторскими стилями*. Стили, определяемые пользовательской таблицей стилей, называются *пользовательскими*, а стили, определяемые браузером, называются *стилями браузера*.

Настройка старшинства важных стилей

Обычную каскадную последовательность просмотра стилей можно изменить, пометив нужные значения свойств как *важные* (листинг 3.11).

Листинг 3.11. Обозначение свойства стиля как важного

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        p {color: black !important;}
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <p style="color: red">Мы рады сообщить о начале предоставления
       новой услуги — доставки заказанных вами цветов прямо на
       дом. В радиусе 20 миль от магазина доставка осуществляется
       бесплатно, доплата за каждую дополнительную милю составляет
       $1.</p>
</body>
</html>
```

Чтобы отметить значения отдельных свойств как важные, следует присоединить к соответствующему объявлению пометку !important. Независимо от того, где определены важные стили, браузер присваивает им наивысший приоритет. Результат задания важности свойств, в данном случае приводящего к тому, что зна-

чение свойства color внедренного стиля перекрывает соответствующее значение встроенного стиля, представлен на рис. 3.10 (хотя заметить отличия на черно-белом рисунке не так-то легко).

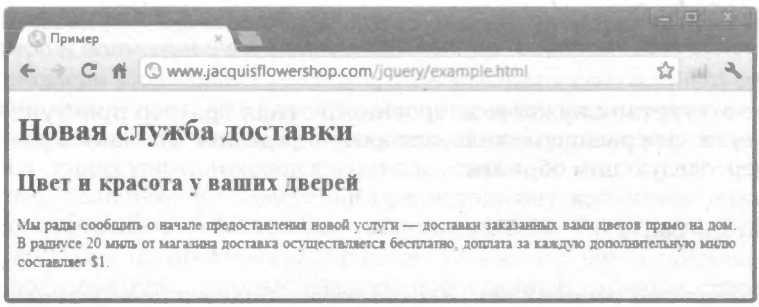


Рис. 3.10. Замена значений свойств встроенных стилей значениями, объявленными как важные

Примечание. Единственное, что может заменить значение свойства стиля, объявленное как важное, — это другое важное значение этого же свойства, объявленное в пользовательской таблице стилей. Обычно авторские стили имеют приоритет по сравнению с пользовательскими стилями, но в случае значений, объявленных как важные, порядок приоритетов меняется на противоположный.

Определение старшинства стилей на основе их специфичности

О возникновении конфликтных ситуаций в процессе назначения стилей элементам говорят в тех случаях, когда к какому-либо элементу могут быть применены два стиля, относящиеся к одному и тому же каскадному уровню и содержащие разные значения CSS-свойства, которое ищет браузер. Чтобы определить, какое именно значение следует использовать, браузер оценивает показатель так называемой *специфичности* каждого стиля и выбирает стиль, являющийся наиболее специфичным. Браузер определяет специфичность стиля, рассчитывая следующие три характеристики:

- количество идентификаторов в селекторе стиля;
- количество других атрибутов, классов и псевдоклассов в селекторе;
- количество элементов и псевдоэлементов в селекторе.

Браузер объединяет значения всех трех оценок для каждого стиля и применяет значение свойства из того стиля, который оказался наиболее специфичным[3]. Простейший пример использования специфичности стилей приведен в листинге 3.12.

Листинг 3.12. Специфичность стилей

```
<!DOCTYPE html>
<html>
```

[3] Любопытно отметить, что унаследованные стили вообще не обладают специфичностью, даже нулевой, и поэтому перекрываются явно заданными стилями любой специфичности (в том числе и нулевой, которой обладает универсальный селектор *). — *Примеч. ред.*

```
<head>
    <title>Пример</title>
    <style>
        p {background-color: grey; color: white;}
        p.details {color: red;}
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <p class="details">Мы рады сообщить о начале предоставления
       новой услуги — доставки заказанных вами цветов прямо на
       дом. В радиусе 20 миль от магазина доставка осуществляется
       бесплатно, доплата за каждую дополнительную милю составляет
       $1.</p>
</body>
</html>
```

При расчете специфичности стиля формируются тройки чисел в виде `a-b-c`, где буквы соответствуют значениям характеристик в том порядке, в каком они были перечислены выше. Это не трехразрядные числа. Более специфичным является стиль, которому соответствует больший показатель специфичности. Сравнение значений `b` производится лишь при равенстве значений `a`. В этом случае более специфичным считается стиль с большим значением `b`. К анализу значений `c` браузер переходит лишь при равенстве пар значений `a-b`. Это означает, что тройке чисел `1-0-0` соответствует большая специфичность, чем тройке `0-5-5`.

В данном случае селектор `p.details` включает в себя атрибут `class`, т.е. специфичность данного стиля оценивается как `0-1-1` (ноль значений `id` плюс один атрибут другого типа плюс одно имя элемента). Специфичность второго селектора оценивается как `0-0-1` (он не включает в себя ни одного значения атрибута `id` или какого-либо другого атрибута и содержит только одно имя элемента).

В процессе визуализации элемента `p` браузер ищет значение свойства `color`. Если элемент `p` относится к классу `details`, то более специфичным будет стиль с селектором, в результате чего для данного элемента `p` будет использоваться значение `red`. Для всех остальных элементов `p` будет использоваться значение `white`. Конечный результат выбора и применения стиля браузером для данного примера показан на рис. 3.11.

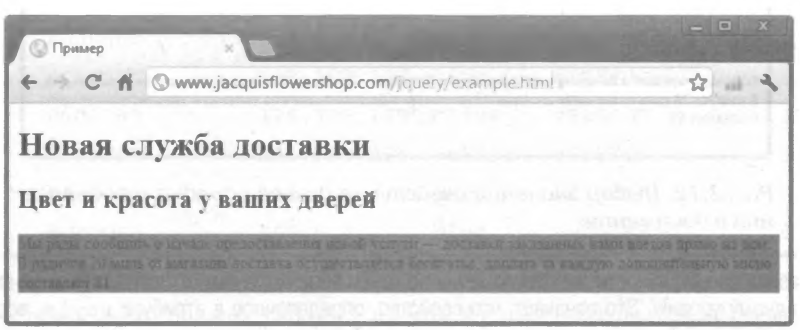


Рис. 3.11. Применение значений свойств стилей на основе их специфичности

Если значения нужных свойств определены одновременно в нескольких стилях с одной и той же специфичностью, то браузер считает наиболее приоритетным то значение, селектор которого определен в документе позже остальных. Пример документа, содержащего два стиля, которые имеют одинаковую специфичность, приведен в листинге 3.13.

Листинг 3.13. Стили с одинаковой специфичностью

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        p.details {color: red;}
        p.information {color: blue;}
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <p class="details information">Мы рады сообщить о начале
       предоставления новой услуги — доставки заказанных вами
       цветов прямо на дом. В радиусе 20 миль от магазина доставка
       осуществляется бесплатно, доплата за каждую дополнительную
       милю составляет $1.</p>
</body>
</html>
```

Оба стиля, определенные в элементе `style`, имеют одинаковые показатели специфичности и оба применяются к элементу `p`. При отображении элемента `p` на данной странице браузер выберет для свойства `color` значение `blue`, поскольку именно оно указано в стиле, который был определен последним. Результат приведен на рис. 3.12.

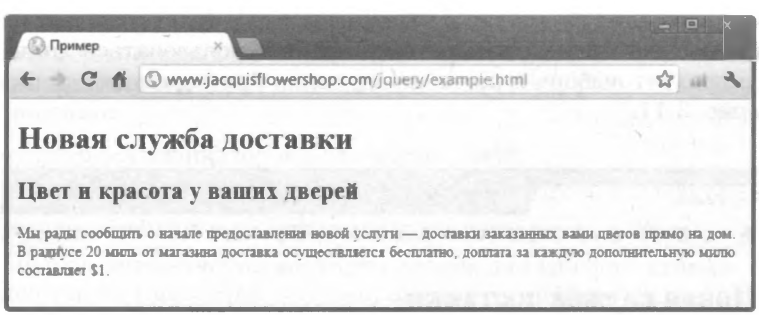


Рис. 3.12. Выбор значений свойств на основе порядка их объявления в документе

Совет. Правила специфичности действуют лишь по отношению к стилям, относящимся к одному и тому же каскадному уровню. Это означает, что свойство, определенное в атрибуте `style`, всегда будет иметь более высокий приоритет по сравнению со свойством, определенным в элементе `style`.

Единицы измерения свойств CSS

Ранее уже приводились значения некоторых свойств CSS, используемые браузером по умолчанию. В примерах эти значения применялись браузером к элементам в соответствии с соглашениями о стилях. Для удобства эта информация повторно приводится в табл. 3.8.

Таблица 3.8. Некоторые свойства CSS и их значения

Свойство	*h1*	*h2*	*p*
color	black	black	black
background-color	transparent	transparent	transparent
font-size	2em	1.5em	16px
border	none	none	none

В CSS определен широкий спектр единиц измерения различного типа. Ниже обсуждаются те из них, которые применяются чаще всего, в том числе те, которые используются в данной книге.

Работа с цветами в CSS

Характеристики цвета используются многими CSS-свойствами, включая свойства color и background-color, которые в данной главе фигурируют во многих примерах. Самый простой способ задания цвета — использование предопределенных названий цветов, но это также можно сделать, указав десятичные или шестнадцатеричные значения для красной, зеленой и синей составляющих. Десятичные значения разделяются запятыми, тогда как шестнадцатеричные значения обычно снабжаются префиксом #, как, например, в записи #ffffff, которая представляет белый цвет. Некоторые из предопределенных названий цветов вместе с соответствующими им десятичными и шестнадцатеричными числовыми эквивалентами приведены в табл. 3.9[4].

Таблица 3.9. Базовые цвета CSS

Название цвета	*Шестнадцатеричный код*	*Десятичный код*	*Название цвета*	*Шестнадцатеричный код*	*Десятичный код*
black	#000000	0,0,0	green	#008000	0,128,0
silver	#C0C0C0	192,192,192	lime	#00FF00	0,255,0
grey	#808080	128,128,128	olive	#808000	128,128,0
white	#FFFFFF	255,255,255	yellow	#FFFF00	255,255,0
maroon	#800000	128,0,0	navy	#000080	0,0,128
red	#FF0000	255,0,0	blue	#0000FF	0,0,255

[4] Чтобы приобрести навыки работы с цветом в различных цветовых схемах, полезно воспользоваться одной из множества доступных утилит, например программой *Color Mania 3.2* (http://www.softsalad.ru/software/colormania.html) или программой *APE Цветик 6.3* (http://www.bestfree.ru/soft/graph/setcolor.php), которые, кроме всего прочего, обеспечивают взятие цветовых проб в любой точке экрана. — *Примеч. ред.*

Окончание табл. 3.9

Название цвета	Шестнадцатеричный код	Десятичный код	Название цвета	Шестнадцатеричный код	Десятичный код
purple	#800080	128,0,128	teal	#008080	0,128,128
fushia	#FF00FF	255,0,255	aqua	#00FFFF	0,255,255

Приведенные цвета называются *базовыми*. В CSS определены также *расширенные* цвета. Их настолько много, что для их перечисления здесь не найдется места. Полный их перечень можно найти по адресу http://www.w3.org/TR/css3-color. Кроме базовых цветов, имеется множество их вариаций. В качестве примера в табл. 3.10 представлен готовый к использованию расширенный набор оттенков серого.

Таблица 3.10. Расширенный набор оттенков серого в CSS

Название цвета	Шестнадцатеричный код	Десятичный код
darkgrey	#a9a9a9	169,169,169
darkslategrey	#2F4F4F	47,79,79
dimgrey	#696969	105,105,105
grey	#808080	128,128,128
lightgrey	#d3d3d3	211,211,211
lightslategrey	#778899	119,136,153
slategrey	#708090	112,128,144

Определение сложных цветов

Использование предопределенных имен или простых шестнадцатеричных кодов не исчерпывает всех возможных способов задания цветов. Для выбора цвета существует ряд функций. Доступные в CSS функции перечислены в табл. 3.11.

Таблица 3.11. Функции для выбора цветов в CSS

Функция	Описание	Пример
rgb(*r*, *g*, *b*)	Определяет цвет в модели RGB	color: rgb(112, 128, 144)
rgba(*r*, *g*, *b*, *a*)	Определяет RGB-цвет с использованием значения альфа-канала для указания степени непрозрачности (0 — полная прозрачность, 1 — полная непрозрачность)	color: rgb(112, 128, 144, 0.4)
hsl(*h*, *s*, *l*)	Определяет цвет путем указания цветового оттенка, насыщенности и яркости (модель HSL)	color: hsl(120, 100%, 22%)
hsla(*h*, *s*, *l*, *a*)	Аналогична функции hsl(), но позволяет дополнительно указать значение альфа-канала, определяющее степень непрозрачности	color: hsla(120, 100%, 22%, 0.4)

Функция rgba() используется для задания цветов, обладающих определенной степенью прозрачности, однако, если вам нужен полностью прозрачный элемент, достаточно использовать специальное значение цвета — transparent.

Единицы измерения длины в CSS

Многие свойства CSS требуют задания *длины*, т.е. линейных размеров объектов. Таковым, например, является свойство `font-size`, с помощью которого можно задать размер шрифта, который должен использоваться для визуализации содержимого элемента. При указании длины ее числовое значение и используемая единица измерения записываются слитно, без пробелов или каких-либо иных символов между ними. Например, запись `20pt` для свойства `font-size` означает `20` единиц, представленных идентификатором `pt` (от *points* — пункты), о которых мы вскоре поговорим. В CSS определены два типа единиц измерения длины: абсолютные и относительные. Относительные единицы определяются по отношению к какому-либо другому свойству. Рассмотрению единиц обоих типов посвящены два следующих раздела.

Абсолютные единицы длины

Абсолютные единицы длины выражают результаты реальных измерений. Спецификация CSS поддерживает пять типов абсолютных единиц измерения, которые описаны в табл. 3.12.

Таблица 3.12. Абсолютные единицы измерения CSS

Единица измерения	*Описание*
`in`	Дюймы
`cm`	Сантиметры
`mm`	Миллиметры
`pt`	Пункты (1 пункт равен 1/72 дюйма)
`pc`	Пики (1 пика равна 12 пунктам)

Допускается смешивание и сочетание разных единиц в одном стиле, а также смешивание абсолютных и относительных единиц. Абсолютные единицы могут быть полезны в тех случаях, когда заранее известно, как будет визуализироваться содержимое (например, выводиться на печать). Могу сказать, что в моих стилях CSS абсолютные единицы встречаются сравнительно редко. Я считаю, что относительные единицы обладают большей гибкостью и с ними проще работать, и поэтому редко создаю содержимое, привязанное к абсолютным величинам.

Примечание. Возможно, вам покажется странным, что в приведенной таблице абсолютных единиц измерения отсутствуют пиксели. В действительности в спецификации CSS делается попытка превратить пиксели в относительную единицу, однако, к сожалению, данная попытка реализована в спецификации весьма небрежно. Более подробно об этом говорится в одном из следующих разделов.

Относительные единицы длины

Задавать и применять относительные единицы сложнее, чем абсолютные, и однозначное определение их смысла требует использования компактного и лаконичного языка. Относительные единицы выражаются через какое-то другие единицы. К сожалению, язык спецификации CSS недостаточно строг (проблема, преследующая CSS на протяжении ряда лет). Здесь имеется в виду, что в CSS определено множество полезных единиц, представляющих интерес, однако некоторые из них невозможно использовать, поскольку они не имеют широкой или унифицирован-

ной браузерной поддержки. В табл. 3.13 приведены относительные единицы, на поддержку которых основными браузерами всегда можно рассчитывать.

Таблица 3.13. Относительные единицы измерения CSS

Единица измерения	*Описание*
`em`	Относительно размера шрифта элемента
`ex`	Относительно высоты символа "x" в элементе
`rem`	Относительно размера шрифта в корневом элементе
`px`	Количество пикселей CSS (в предположении, что содержимое отображается на экране с разрешением 96 dpi)
`%`	Процентная доля от значения другого свойства

Фактически, используя относительные единицы, вы выражаете одну величину через результаты измерения другой. Пример задания значения свойства через значение свойства `font-size` приведен в листинге 3.14.

Листинг 3.14. Использование относительных единиц

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        p.details {
            font-size: 15pt;
            height: 3.5em;
            border: thin solid black;
        }
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <p class="details information">Мы рады сообщить о начале
       предоставления новой услуги – доставки заказанных вами
       цветов прямо на дом. В радиусе 20 миль от магазина доставка
       осуществляется бесплатно, доплата за каждую дополнительную
       милю составляет $1.</p>
</body>
</html>
```

В этом примере значение свойства `height` (определяющее высоту элемента) устанавливается равным `3.5em`. Это означает, что при визуализации элементов `p` на экране их высота будет в три с половиной раза превышать значение `font-size`. В этом легко убедиться, взглянув на рис. 3.13. Чтобы облегчить визуальное сравнение высоты элемента `p` с размером шрифта, элемент окружен рамкой.

Работа с пикселями

Пиксели в CSS — это не то, чего можно было бы ожидать. Обычно под термином *пиксель* понимается наименьший адресуемый элемент изображения на экране. Спецификация CSS пытается придать этому термину иное значение и определяет его следующим образом.

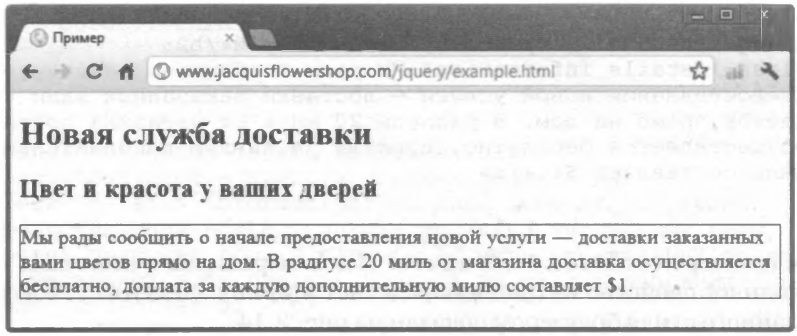


Рис. 3.13. Результат использования относительных единиц измерения

Эталонный пиксель — это зрительный угол, под которым виден один пиксель на устройстве с плотностью пикселей 96 dpi при условии, что он рассматривается с расстояния вытянутой руки.

Перед вами типичный пример одного из тех туманных определений, которые сплошь и рядом встречаются в спецификации CSS. Я не собираюсь долго рассуждать на эту тему и лишь замечу, что спецификации, в которых определения основаны на "длине руки пользователя", не вызывают у меня особого доверия. К счастью, основные модели браузеров игнорируют различия между пикселями, определенными в CSS, и пикселями на экране и трактуют один пиксель как 1/96 дюйма. (Принятая здесь плотность пикселей является стандартной в Windows; на других платформах с экранах, плотность пикселей которых отличается от указанной, браузеры обычно выполняют соответствующий перерасчет, так что и в этих случаях один пиксель по-прежнему составляет примерно 1/96 дюйма.)

Примечание. Хотя большой пользы это вам не принесет, можете прочитать полное определение пикселя CSS, которое находится по следующему адресу:

`http://www.w3.org/TR/CSS21/syndata.html#length-units`

В конечном счете, несмотря на то что пиксели CSS задумывались в качестве относительной единицы измерения, на практике браузеры интерпретируют их как абсолютные единицы. Пример использования пикселей в CSS-стиле приведен в листинге 3.15.

Листинг 3.15. Использование пикселей в качестве единицы измерения

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        p.details {
            font-size: 20px;
            width: 400px;
            border: thin solid black;
        }
    </style>
</head>
<body>
```

```
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <p class="details information">Мы рады сообщить о начале
       предоставления новой услуги — доставки заказанных вами
       цветов прямо на дом. В радиусе 20 миль от магазина доставка
       осуществляется бесплатно, доплата за каждую дополнительную
       милю составляет $1.</p>
</body>
</html>
```

В этом примере свойства `font-size` и `width` выражены в пикселях (свойство `width` дополняет свойство `height` и определяет ширину элемента.) Результат применения данного стиля браузером показан на рис. 3.14.

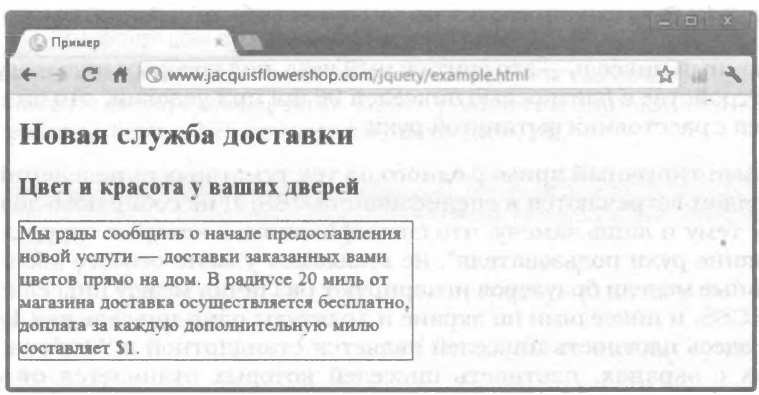


Рис. 3.14. Результат указания размера элемента в пикселях

Совет. Я часто использую пиксели для выражения значений различных величин в CSS, однако это, скорее всего, результат привычки. С моей точки зрения, единицы `em` обеспечивают большую гибкость. Объясняется это очень просто: если мне необходимо внести изменения, то в этом случае достаточно изменить лишь размер шрифта, и вся остальная часть стиля по-прежнему будет работать безукоризненно. Важно не забывать, что с практической точки зрения пиксели CSS являются абсолютными единицами и в силу этого могут не обеспечивать требуемую гибкость.

Работа с процентными значениями

В качестве единиц измерения свойств могут использоваться процентные доли от значений других свойств. В этом случае следует использовать символ `%`, как показано в листинге 3.16.

Листинг 3.16. Выражение значения одного свойства через процентную долю значения другого свойства

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <style>
        p.details {
            font-size: 200%;
            width: 50%;
```

```
            border: thin solid black;
        }
    </style>
</head>
<body>
    <h1 lang="ru">Новая служба доставки</h1>
    <h2 lang="ru">Цвет и красота у ваших дверей</h2>
    <p class="details information">Мы рады сообщить о начале
       предоставления новой услуги — доставки заказанных вами
       цветов прямо на дом. В радиусе 20 миль от магазина доставка
       осуществляется бесплатно, доплата за каждую дополнительную
       милю составляет $1.</p>
</body>
</html>
```

При использовании процентов возникают две сложности. Первая из них обусловлена тем, что не все свойства могут быть выражены таким способом, а вторая — тем, что каждое свойство, которое *может* быть выражено процентным значением, самостоятельно определяет, по отношению к какому именно *другому* свойству это значение определяется. Например, свойство `font-size` использует значение `font-size`, наследуемое от родительского элемента, а свойство `width` использует значение свойства `width` элемента-контейнера.

Использование сокращенных свойств и специальных значений

Задание размерных величин или значений цветов требуется не для всех свойств. Некоторые из них имеют специальные значения, уникальные для тех аспектов поведения элементов, которыми они управляют. В качестве примера можно привести свойство `border`, которое использовалось в ряде примеров для окружения элементов рамками. Свойство `border` позволяет задать одновременно три значения свойств границы с использованием специальных значений, как показано ниже.

```
border: thin solid black
```

Первое значение определяет толщину линий, второе — их стиль, а третье — цвет. Возможные способы задания толщины линий границы перечислены в табл. 3.14.

Таблица 3.14. Способы задания толщины границы

Значение	*Описание*
<толщина>	Толщина линий границы, выраженная в единицах CSS, таких как `em`, `px` или `cm`
<проценты>`%`	Процентная доля значения *ширины* области, для которой определяется граница

`thin`
`medium`
`thick`
	Предустановленные значения толщины границы, увеличивающиеся в приведенной последовательности, точная величина которых определяется браузером

Допустимые значения стиля линий приведены в табл. 3.15.

Путем сочетания значений из этих таблиц со значениями цвета можно изменять внешний вид границ в широких пределах. Пример того, как различные стили границы отображаются в браузере, приведен на рис. 3.15.

Таблица 3.15. Значения стиля границы

Значение	*Описание*
`none`	Граница отсутствует
`dashed`	Граница, образованная серией прямоугольников
`dotted`	Граница, образованная серией круглых точек
`double`	Граница, образованная двумя линиями, разделенными промежутком
`groove`	Граница выглядит так, будто она вдавлена в страницу
`inset`	Граница выглядит так, будто содержимое вдавлено в страницу
`outset`	Граница выглядит так, будто содержимое выступает над страницей
`ridge`	Граница выглядит так, будто выступает над страницей
`solid`	Граница в виде сплошной одинарной линии

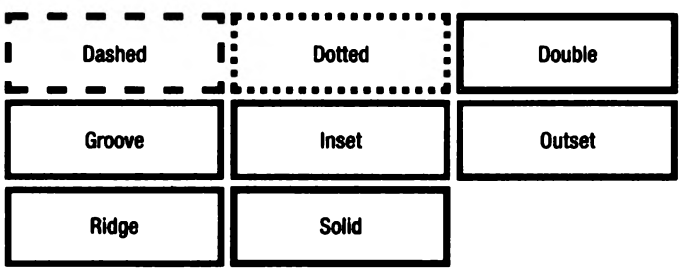


Рис. 3.15. Стили границ

Кроме того, свойство `border` является ярким примером так называемых *сокращенных свойств* (shorthand properties). К таковым относятся свойства, с помощью которых можно задать сразу весь набор значений для целой группы взаимосвязанных свойств в одном объявлении. Например, объявление свойства `border` в том виде, в каком оно использовалось в приведенных выше примерах, эквивалентно 12 объявлениям, приведенным в листинге 3.17.

Листинг 3.17. Индивидуальные свойства `border`

```
border-top-color: black;
border-top-style: solid;
border-top-width: thin;
border-bottom-color: black;
border-bottom-style: solid;
border-bottom-width: thin;
border-left-color: black;
border-left-style: solid;
border-left-width: thin;
border-right-color: black;
border-right-style: solid;
border-right-width: thin;
```

CSS предоставляет возможность либо детализировать свойства и более точно контролировать внешний вид границ, задавая значения отдельных свойств, либо

использовать сокращенные свойства в тех случаях, когда значения родственных свойств совпадают между собой.

Резюме

В этой главе был дан краткий обзор спецификации CSS и было показано, как задавать значения свойств с помощью атрибута `style`, как использовать элемент `style` (включая широкий ряд доступных селекторов) и как браузеры, используя каскадирование и специфичность стилей, определяют, какие значения свойств должны использоваться при отображении элементов. В завершение было кратко рассказано об используемых в CSS единицах измерения, а также о специальных значениях свойств и сокращенных свойствах, обозначающих в краткой записи целую группу свойств. Для выражения значений свойств существует ряд различных способов, что придает стилям CSS дополнительную гибкость.

ГЛАВА 4

Введение в JavaScript

Библиотека jQuery — это набор функций, оптимизирующих доступ к элементам HTML и их атрибутам с помощью JavaScript. Чтобы воспользоваться функциональностью jQuery, вы должны добавить в HTML-документ не только эту библиотеку, но и собственные сценарии JavaScript. Данная глава служит кратким введением в JavaScript, в котором основное внимание уделено возможностям языка, применяемым для работы с jQuery.

Язык JavaScript прошел долгий путь развития и получил возможность окончательно оформиться лишь после того, как был стандартизирован. Работа с JavaScript доставит вам удовольствие, однако потребует знания некоторых его особенностей.

Совет. Максимальную пользу эта книга принесет тем читателям, которые имеют хотя бы некоторый опыт программирования и чувствуют себя уверенно при обсуждении таких понятий, как переменные, функции и объекты. Если же вы новичок в программировании, то можете начать с чтения серии статей, опубликованных на сайте Lifehacker.com. В них не предполагается, что читатель обладает навыками программирования, а все примеры, что немаловажно, написаны на языке JavaScript. Указанное руководство доступно по следующему адресу:

`http://lifehacker.com/5744113/learn-to-code-the-full-beginners-guide`

Знакомство с JavaScript

Для включения JavaScript-сценариев в HTML-документы существует несколько способов. Можно определить *встроенный сценарий*, содержимое которого является частью документа. Также можно определить *внешний сценарий*, JavaScript-код которого хранится в отдельном файле, тогда как HTML-документ лишь ссылается на него посредством URL-адреса (как вы увидите в главе 5, именно так осуществляется доступ к библиотеке jQuery). Оба подхода основаны на использовании элемента `script`. В этой главе мы будем пользоваться встроенными сценариями. Пример простого сценария приведен в листинге 4.1.

Листинг 4.1. Простой встроенный сценарий

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        console.log("Привет");
    </script>
</head>
```

```
<body>
    Это простой пример
</body>
</html>
```

Этот чрезвычайно простой сценарий всего лишь выводит сообщение на *консоль*. Консоль — это простое (но весьма полезное) средство, предоставляемое браузером и предназначенное для отображения отладочной информации, генерируемой сценарием в процессе выполнения. Вызов консоли осуществляется в разных браузерах по-разному. В Google Chrome для этого следует выбрать пункт Консоль JavaScript в меню Инструменты. Вид консоли, отображаемой в окне браузера Google Chrome, приведен на рис. 4.1; аналогичные средства предусмотрены и в других браузерах.

Рис. 4.1. Окно консоли JavaScript в браузере Google Chrome

Как нетрудно заметить, результат, генерируемый вызовом метода `console.log()`, отображается в окне консоли вместе с дополнительной информацией о местонахождении источника сообщения (в данном случае это строка 6 файла `example.com`). В данной главе вместо копий экрана будут приводиться лишь результаты выполнения примеров, выводимые на консоль. Представленный выше сценарий генерирует следующий результат.

```
Привет
```

Чтобы облегчить чтение результатов, некоторые из них будут представляться в предварительно отформатированном виде. В следующих разделах демонстрируются основные возможности языка JavaScript. Если у вас уже имеется опыт программирования с использованием какого-либо другого современного языка, то никаких затруднений при изучении синтаксиса JavaScript у вас возникать не должно.

Использование инструкций

Основными строительными кирпичиками JavaScript являются *инструкции*. Каждая инструкция представляет одну команду и обычно заканчивается точкой с запятой (`;`). В действительности наличие этого символа не является обязательным,

однако его использование облегчает чтение кода и позволяет записывать несколько инструкций в одной строке. В листинге 4.2 приведен пример сценария с двумя инструкциями, добавленными в документ с помощью элемента `script`.

Листинг 4.2. Использование инструкций JavaScript

```
<!DOCTYPE html>
<html>
    <head>
        <title>Пример</title>
    </head>
    <body>
        <script type="text/javascript">
            console.log("Это инструкция");
            console.log("Это тоже инструкция");
        </script>
    </body>
</html>
```

Браузер последовательно выполняет одну инструкцию сценария за другой. В данном случае на консоль будет выведен следующий результат.

```
Это инструкция
Это тоже инструкция
```

Определение и использование функций

Если инструкции определяются непосредственно в элементе `script`, как это было сделано в листинге 4.2, то браузер выполнит эти инструкции, как только достигнет их в процессе загрузки документа. Другой возможный вариант — поместить несколько инструкций в *функцию*, которая не будет выполняться до тех пор, пока браузер не встретит инструкцию, вызывающую эту функцию, как показано в листинге 4.3.

Листинг 4.3. Определение функции JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        function myFunc() {
            console.log("Это инструкция");
        };

        myFunc();
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Инструкции, которые содержатся внутри функции, заключаются в фигурные скобки и называются *блоком кода*. В приведенном выше листинге определена

функция с именем myFunc, блок кода которой состоит всего из одной инструкции. JavaScript — чувствительный к регистру символов язык, поэтому ключевое слово function должно быть набрано строчными буквами. Содержащаяся в этой функции инструкция не будет выполнена до тех пор, пока браузер не достигнет другой инструкции, содержащей вызов функции myFunc(). В данном случае таковой является следующая инструкция:

```
myFunc();
```

Приведенный пример не очень показателен, поскольку вызов функции следует сразу же за ее определением. С некоторыми примерами, в которых функции играют гораздо более важную роль, вы познакомитесь, когда мы будем рассматривать события в главе 9.

Определение функций с параметрами

Аналогично большинству других языков программирования, JavaScript позволяет определять функции, имеющие *параметры*, как показано в листинге 4.4.

Листинг 4.4. Определение функции, имеющей параметры

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        function myFunc(name, weather) {
            console.log("Привет, " + name + "!");
            console.log("Сегодня " + weather + " день");
        };

        myFunc("Адам", "солнечный");
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом листинге в функцию myFunc() добавлены два параметра с именами name и weather. JavaScript — слабо типизированный язык. Это означает, что указывать типы параметров в определении функции не требуется. К вопросу о слабой типизации мы еще вернемся, когда будем рассматривать переменные JavaScript. Конкретные значения параметров передаются функции при ее вызове в виде *аргументов*.

```
myFunc("Адам", "солнечный");
```

Данный сценарий выведет на консоль следующий результат.

```
Привет, Адам!
Сегодня солнечный день
```

Число аргументов, указываемых при вызове функции, может не совпадать с числом параметров в определении функции. Если число аргументов, указанных при вызове функции, меньше того их числа, которое предусмотрено определением функции, то пропущенные параметры будут иметь неопределенные значения. Если

же число указанных аргументов превышает число параметров функции, то избыточные аргументы просто игнорируются.

Отсюда следует, что одновременное присутствие в сценариях JavaScript двух одноименных функций невозможно, поскольку различие в числе параметров, указанных в их определениях, не может быть использовано в качестве отличительного признака. Подобная возможность, известная как *полиморфизм функций*, поддерживается в ряде языков программирования (например, Java или C#), однако в JavaScript она недоступна. Вместо этого, если вы одновременно определите две одноименные функции, второе определение вытеснит первое.

Определение функций, возвращающих результат

Функции могут возвращать результат с помощью ключевого слова `return`. Пример такой функции приведен в листинге 4.5.

Листинг 4.5. Возвращение результата функцией

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        function myFunc(name) {
            return("Привет, " + name + "!");
        };

        console.log(myFunc("Адам"));
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Данная функция имеет один параметр, который используется для получения простого результата. Возвращаемый функцией результат передается функции `console.log()` в качестве аргумента.

```
console.log(myFunc("Адам"));
```

Обратите внимание: никакого объявления о том, что функция будет возвращать результат, равно как и указания типа возвращаемого результата, не требуется. Данный сценарий выведет на консоль следующий результат

```
Привет, Адам!
```

Использование переменных и типов

Для объявления переменных используется ключевое слово `var`. Одновременно с объявлением переменной ее можно инициализировать некоторым значением. Переменные, объявленные в функции, называются *локальными переменными*, и их можно использовать только внутри этой функции. Переменные, объявленные внутри самого элемента `<script>`, называются *глобальными переменными*. Доступ к ним может осуществляться из любого места в документе, в том числе из других

сценариев. Пример использования локальных и глобальных переменных приведен в листинге 4.6.

Листинг 4.6. Использование локальных и глобальных переменных

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Пример</title>
    </head>
    <body>
        <script type="text/javascript">
            var myGlobalVar = "яблоки";

            function myFunc(name) {
                var myLocalVar = "солнечный";
                return ("Привет, " + name + "! Сегодня " +
                myLocalVar + " день.");
            };
            document.writeln(myFunc("Адам"));
        </script>
        <script type="text/javascript">
            document.writeln("Я люблю " + myGlobalVar);
        </script>
    </body>
</html>
```

Вновь подчеркну, что JavaScript — слабо типизированный язык. Отсюда вовсе не следует, что типы в JavaScript вообще отсутствуют. Это лишь означает, что объявлять тип переменных не требуется и можно свободно присваивать одной и той же переменной различные типы значений. JavaScript самостоятельно определяет типы переменных, исходя из присвоенных им значений, и выполняет преобразование типов в зависимости от того, в каком контексте они используются. Результат работы данного сценария, отображаемый в окне браузера, будет выглядеть следующим образом.

```
Привет, Адам! Сегодня солнечный день.
Я люблю яблоки
```

Использование примитивных типов

В JavaScript определен небольшой набор примитивных типов данных. К ним относятся типы данных `string`, `number` и `boolean`. Этот список кажется совсем коротким, но JavaScript умудряется придать данной тройке типов значительную гибкость.

Работа со строками

Для определения строк, образующих тип `string`, используются двойные или одинарные кавычки, как показано в листинге 4.7.

Листинг 4.7. Объявление строковых переменных

```
<!DOCTYPE HTML>
<html>
<head>
```

```
    <title>Пример</title>
    <script type="text/javascript">
        var firstString = "Это строка";
        var secondString = 'Это тоже строка';
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В каждой паре кавычки должны быть одного типа. Например, нельзя начать объявление строки с одинарной кавычки, а закончить — двойной.

Работа с булевыми переменными

Логические переменные, т.е. переменные типа `boolean`, могут принимать одно из двух литеральных значений: `true` или `false`. Пример использования логических (булевых) переменных представлен в листинге 4.8, однако чаще всего переменные этого типа используются в условных выражениях, таких как инструкция `if`.

Листинг 4.8. Объявление булевых переменных

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var firstBool = true;
        var secondBool = false;
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Работа с числами

Тип `number` используется для представления как *целых* чисел, так и чисел *с плавающей запятой* (точкой), называемых также *действительными* числами. Соответствующий пример приведен в листинге 4.9.

Листинг 4.9. Объявление числовых переменных

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var daysInWeek = 7;
        var pi = 3.14;
        var hexValue = 0xFFFF;
    </script>
</head>
<body>
```

```
    Это простой пример
</body>
</html>
```

Указывать, какой тип чисел вы собираетесь использовать, не требуется. Вы просто записываете нужное значение, и JavaScript самостоятельно решает, что с ним делать. В данном листинге определены: целочисленная переменная, переменная с плавающей точкой и шестнадцатеричная переменная, значение которой снабжено префиксом `0x`.

Создание объектов

JavaScript поддерживает объекты и обеспечивает несколько способов их создания. Соответствующий пример приведен в листинге 4.10.

Листинг 4.10. Создание объекта

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = new Object();
        myData.name = "Адам";
        myData.weather = "солнечный";

        console.log("Привет, " + myData.name + "! ");
        console.log("Сегодня " + myData.weather + " день.");
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В данном сценарии объект создается посредством вызова `new Object()`, а полученный результат (вновь созданный объект) присваивается переменной `myData`. Создав объект, мы можем определять для него свойства и присваивать им значения.

```
myData.name = "Адам";
```

До выполнения этого оператора свойство `name` у объекта `myData` отсутствовало. После выполнения этого оператора оно уже существует и содержит значение `Адам`. Для обращения к значению свойства следует присоединить имя свойства к имени переменной с помощью точки (`.`).

```
console.log("Привет, " + myData.name + "! ");
```

Использование объектных литералов

Свойства объекта можно также определить во время его создания, используя формат *объектного литерала*. Пример того, как это делается, приведен в листинге 4.11.

Листинг 4.11. Использование формата объектного литерала

```
<!DOCTYPE HTML>
<html>
<head>
```

```
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный"
        };

        console.log("Привет, " + myData.name + "! ");
        console.log("Сегодня " + myData.weather + " день.");
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Каждое присваиваемое значение записывается справа от имени свойства и отделяется от него двоеточием (:), а в качестве разделителя между парами "свойство–значение" используется запятая (,).

Использование функций в качестве методов

С моей точки зрения, одной из наиболее привлекательных возможностей языка JavaScript является способ, которым в объект добавляются функции. Объявленная в объекте функция называется *методом*. Пример добавления метода в объект приведен в листинге 4.12.

Листинг 4.12. Добавление методов в объект

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
            printMessages: function() {
                console.log("Привет, " + this.name + "! ");
                console.log("Сегодня " + this.weather + " день.");
            }
        };
        myData.printMessages();
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом сценарии функция используется для создания метода под названием `PrintMessages`. Обратите внимание, что для ссылки на свойства объекта используется ключевое слово `this`. Если функция используется в качестве метода, то объект, для которого вызывается данный метод, передается этой функции в виде аргумента через специальную переменную `this`. Результат выполнения этого сценария будет выглядеть следующим образом.

```
Привет, Адам!
Сегодня солнечный день.
```

Работа с объектами

Создав нужные объекты, вы сможете выполнять над ними различные операции. В следующих разделах описываются те операции, которые встречаются в книге.

Получение и изменение значений свойств

Простейшие операции, которые можно выполнять над объектами, — это чтение и изменение значений определяемых ими свойств. Существует два стилистически различных синтаксиса, которые можно использовать для этой цели (листинг 4.13).

Листинг 4.13. Получение и изменение свойств объекта

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
        };

        myData.name = "Джо";
        myData["weather"] = "дождливый";

        console.log("Привет, " + myData.name + "!");
        console.log("Сегодня " + myData["weather"] + " день.");
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Первый стиль — так называемая *точечная нотация* — должен быть знаком большинству программистов, и именно он использовался в приведенных ранее примерах. При использовании этого стиля имя свойства присоединяется к имени объекта с помощью точки (.).

```
myData.name = "Джо";
```

Второй стиль предполагает использование индексов, как при работе с массивами.

```
myData["weather"] = "дождливый";
```

В этом случае имя свойства заключается в квадратные скобки ([и]). Этот способ весьма удобен для доступа к свойствам, поскольку он позволяет задавать имя свойства с помощью переменной.

```
var myData = {
    name: "Адам",
    weather: "солнечный",
```

```
};

var propName = "weather";
myData[propName] = "дождливый";
```

Именно этот способ используется ниже для циклического перебора свойств объектов.

Просмотр свойств объектов

Для просмотра свойств объекта используется блок `for...in`. Пример использования этого цикла приведен в листинге 4.14.

Листинг 4.14. Просмотр свойств объекта

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
            printMessages: function() {
                console.log("Привет, " + this.name + "! ");
                console.log("Сегодня " + this.weather + " день.");
            }
        };

        for (var prop in myData) {
            console.log("Имя: " + prop + " Значение: " +
            myData[prop]);
        }

    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Цикл `for...in` обеспечивает выполнение инструкции, содержащейся внутри блока кода, для каждого свойства объекта `myData`. Переменной `prop` присваивается имя свойства, обрабатываемого во время текущей итерации. Доступ к значениям отдельных свойств осуществляется с использованием индексов в стиле массивов. Результат работы данного сценария будет выглядеть так.

```
Имя: name Значение: Адам
Имя: weather Значение: солнечный
Имя: printMessages Значение: function () {
    console.log("Привет, " + this.name + "!");
    console.log("Сегодня " + this.weather + " день.");
}
```

Как нетрудно заметить, среди выводимых значений свойств находится также функция, определенная в объекте в качестве метода. Это лишний раз демонстрирует ту гибкость, которую JavaScript допускает в обращении с функциями.

Добавление и удаление свойств и методов

Даже если объект был создан с использованием стиля объектного литерала, ничто не мешает определять для него новые свойства, как показано в листинге 4.15.

Листинг 4.15. Добавление нового свойства в объект

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
        };

        myData.dayOfWeek = "Понедельник";
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом сценарии в объект добавляется новое свойство — `dayOfWeek`. В данном случае использована точечная нотация (конкатенация имен объекта и свойства с помощью точки), но с равным правом здесь можно было бы использовать индексную нотацию.

Как вы уже, наверное, догадались, в объект можно добавлять не только новые свойства, но и новые методы, указывая функцию в качестве значения нового свойства (листинг 4.16).

Листинг 4.16. Добавление нового метода в объект

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
        };

        myData.SayHello = function() {
          console.write("Привет, ");
        };
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Для удаления свойства или метода используется ключевое слово `delete`, как показано в листинге 4.17.

Листинг 4.17. Удаление свойства из объекта

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
        };

        delete myData.name;
        delete myData["weather"];
        delete myData.SayHello;
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Проверка наличия свойства у объекта

Можно проверить, имеет ли объект интересующее вас свойство, используя выражение `in`, как показано в листинге 4.18.

Листинг 4.18. Проверка того, имеет ли объект указанное свойство

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
        };

        var hasName = "name" in myData;
        var hasDate = "date" in myData;

        console.log("HasName: " + hasName);
        console.log("HasDate: " + hasDate);
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом примере тестируется как имеющееся, так и отсутствующее свойство. Значением переменной `hasName` будет `true`, а значением переменной `hasDate` — `false`.

Использование операторов JavaScript

Определяемый в JavaScript набор допустимых операторов в целом является стандартным. Наиболее употребительные из них приведены в табл. 4.1.

Таблица 4.1. Наиболее употребительные операторы JavaScript

Операторы	*Описание*
`++, --`	Операции инкремента и декремента (каждая из них может быть как префиксной, так и постфиксной)
`+, -, *, /, %`	Сложение, вычитание, умножение, деление, деление по модулю (остаток от деления)
`<, <=, >, >=`	Меньше чем, меньше или равно, больше чем, больше или равно
`==, !=`	Равно, не равно
`===, !==`	Тождественно, не тождественно
`&&, ||`	Логические И и ИЛИ
`=`	Операция присваивания
`+`	Конкатенация строк
`?:`	Тернарная (тройная) условная операция

Использование условных операторов

Многие операции JavaScript используются в сочетании с условными операторами. В этой книге будут часто использоваться блоки `if/else` и `switch`. Пример использования этих инструкций приведен в листинге 4.19 (тем, кто уже занимался программированием, все это должно быть знакомо).

Листинг 4.19. Использование блоков `if/else` и `switch`

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var name = "Адам";

        if (name == "Адам") {
            console.log("Имя - Адам");
        } else if (name == "Джеки") {
            console.log("Имя - Джеки");
        } else {
            console.log("Имя - ни Адам, ни Джеки");
        }

        switch (name) {
            case "Адам":
                console.log("Имя - Адам");
                break;
            case "Джеки":
```

```
                console.log("Имя - Джеки");
                break;
            default:
                console.log("Имя - ни Адам, ни Джеки");
                break;
        }
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Логические операции равенства и тождественности

Логические операции равенства и тождественности (строгого равенства) заслуживают отдельного рассмотрения. Операция равенства всегда пытается привести операнды к одному и тому же типу, чтобы иметь возможность сравнить их между собой. Соответствующий пример приведен в листинге 4.20.

Листинг 4.20. Использование логической операции равенства

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

       var firstVal = 5;
       var secondVal = "5";

        if (firstVal == secondVal) {
            console.log("Значения совпадают");
        } else {
            console.log("Значения НЕ совпадают");
        }
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Результат будет выглядеть так.

```
Значения совпадают
```

JavaScript приводит оба операнда к одному типу и выполняет их сравнение. По сути, операция равенства тестирует равенство значений двух величин независимо от их типа. Если же вы хотите убедиться в том, что совпадают не только значения, но и типы сравниваемых данных, то следует использовать операцию тождественного, или строгого, равенства (===, три знака равенства, а не два, в отличие от операции равенства), как показано в листинге 4.21.

Листинг 4.21. Использование операции тождественного равенства

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

       var firstVal = 5;
       var secondVal = "5";

        if (firstVal === secondVal) {
            console.log("Значения совпадают");
        } else {
            console.log("Значения НЕ совпадают");
        }
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом примере операция тождественного равенства определит, что сравниваемые переменные не совпадают между собой, поскольку принудительное приведение типов в данном случае не выполняется. Результат будет выглядеть так.

```
Значения НЕ совпадают
```

Примитивы JavaScript сравниваются по значениям, но объекты JavaScript сравниваются по ссылкам. Пример того, как JavaScript справляется с тестированием равенства и тождественности двух объектов, приведен в листинге 4.22.

Листинг 4.22. Тестирование равенства и тождественности объектов

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var myData1 = {
            name: "Адам",
            weather: "солнечный",
        };

        var myData2 = {
            name: "Адам",
            weather: "солнечный",
        };

        var myData3 = myData2;

        var test1 = myData1 == myData2;
        var test2 = myData2 == myData3;
        var test3 = myData1 === myData2;
        var test4 = myData2 === myData3;
```

```
        console.log("Тест 1: " + test1 + " Тест 2: " + test2);
        console.log("Тест 3: " + test3 + " Тест 4: " + test4);
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Результат будет выглядеть так.

```
Тест 1: false Тест 2: true
Тест 3: false Тест 4: true
```

В листинге 4.23 приведен сценарий, который выполняет те же тесты, но по отношению к примитивам.

Листинг 4.23. Тестирование равенства и тождественности примитивов

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var myData1 = 5;
        var myData2 = "5";
        var myData3 = myData2;

        var test1 = myData1 == myData2;
        var test2 = myData2 == myData3;
        var test3 = myData1 === myData2;
        var test4 = myData2 === myData3;

        console.log("Тест 1: " + test1 + " Тест 2: " + test2);
        console.log("Тест 3: " + test3 + " Тест 4: " + test4);
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Результат будет выглядеть так.

```
Тест 1: true Тест 2: true
Тест 3: false Тест 4: true
```

Явное преобразование типов

Операция конкатенации строк (+) имеет более высокий приоритет по сравнению с операцией сложения (также +), откуда следует, что JavaScript будет пытаться сначала конкатенировать переменные и лишь затем складывать их. Это может вносить путаницу, поскольку в процессе вычисления результата JavaScript будет

самостоятельно преобразовывать типы, и этот результат не всегда будет соответствовать вашим ожиданиям, как показано в листинге 4.24.

Листинг 4.24. Приоритет операции конкатенации строк

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var myData1 = 5 + 5;
        var myData2 = 5 + "5";

        console.log("Результат 1: " + myData1);
        console.log("Результат 2: " + myData2);

    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Результат будет выглядеть так.

```
Результат 1: 10
Результат 2: 55
```

Многим второй результат может показаться странным. Чрезмерный энтузиазм JavaScript в отношении преобразования типов в сочетании с необходимостью учета старшинства операций привел к тому, что оператор сложения был интерпретирован как оператор конкатенации строк. Чтобы этого избежать, вам иногда придется явно преобразовывать типы значений, тем самым обеспечивая нужный порядок выполнения операций.

Преобразование чисел в строки

Если вы работаете с несколькими числовыми переменными и хотите конкатенировать их в виде строк, то в этом случае следует использовать метод `toString()`, как показано в листинге 4.25.

Листинг 4.25. Использование метода `number.toString()`

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData1 = (5).toString() + String(5);
        console.log("Результат: " + myData1);
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Обратите внимание на то, что вызов метода `toString()` выполняется для числового значения, предварительно заключенного в скобки. Это необходимо было сделать для того, чтобы позволить JavaScript преобразовать литеральное значение в тип `number`, прежде чем вызывать метод, определенный для этого типа. В этом же листинге демонстрируется другой возможный подход, обеспечивающий получение того же результата, который заключается в вызове функции `String()` с передачей ей числового значения в качестве аргумента. Результат будет одним и тем же в обоих случаях: значение типа `number` преобразуется в значение типа `string`, т.е. операция + будет использоваться для выполнения конкатенации строк, а не для сложения. Сценарий, приведенный в листинге 4.25, даст следующий результат.

```
Результат: 55
```

Существует ряд других методов, позволяющих более точно управлять представлением чисел в виде строк. Их краткое описание приведено в табл. 4.2. Все перечисленные методы определены для типа `number`.

Таблица 4.2. Полезные методы для преобразования чисел в строки

Метод	*Описание*	*Тип возвращаемого значения*
`toString()`	Представляет число в десятичной системе счисления	`string`

`toString(2)`
`toString(8)`
`toString(16)`
	Представляет число в двоичном, восьмеричном или шестнадцатеричном виде	`string`

`toFixed(n)`	Представляет действительное число с *n* значащими цифрами после десятичной точки	`string`
`toExponential(n)`	Представляет число в экспоненциальной форме с одной цифрой перед десятичной точкой и *n* цифрами после нее	`string`
`toPrecision(n)`	Представляет число с *n* значащими цифрами, используя экспоненциальную форму в случае необходимости	`string`

Преобразование строк в числа

Противоположный характер носит задача преобразования строк в числа для последующего выполнения операции сложения чисел, а не конкатенации строк. Для этого предусмотрена функция `Number()`, пример использования которой приведен в листинге 4.26.

Листинг 4.26. Преобразование строк в числа

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var firstVal = "5";
        var secondVal = "5";

        var result = Number(firstVal) + Number(secondVal);
```

```
        console.log("Результат: " + result);
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Результат для листинга 4.26 будет выглядеть так.

```
Результат: 10
```

Метод `Number()` довольно строг в отношении синтаксического анализа строковых значений, однако существуют две другие функции, обладающие большей гибкостью, которые игнорируют замыкающие символы, не являющиеся цифрами. Это функции `parseInt()` и `parseFloat()`. Все три метода описаны в табл. 4.3.

Таблица 4.3. Полезные методы для преобразования строк в числа

Метод	*Описание*
`Number(`*строка*`)`	Выполняет синтаксический анализ указанной строки для создания целого или действительного значения
`parseInt(`*строка*`)`	Выполняет синтаксический анализ указанной строки для создания целого значения
`parseFloat(`*строка*`)`	Выполняет синтаксический анализ указанной строки для создания целого или действительного значения

Работа с массивами

Массивы JavaScript работают почти так же, как и массивы в большинстве других языков программирования. Пример создания массива и заполнения его значениями приведен в листинге 4.27.

Листинг 4.27. Создание массива и заполнение его значениями

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var myArray = new Array();
        myArray[0] = 100;
        myArray[1] = "Адам";
        myArray[2] = true;

    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом листинге посредством вызова `new Array()` создается пустой массив `myArray`. Последующие инструкции заполняют значениями элементы этого массива, доступ к которым осуществляется с использованием индексов.

В отношении этого примера можно сделать два замечания. Во-первых, при создании массива не требовалось объявлять, сколько элементов он будет содержать. В JavaScript размер массива автоматически увеличивается при заполнении его значениями, так что он может содержать любое число элементов. Во-вторых, не требовалось объявлять и типы данных, которые будут содержаться в массиве. Любой массив JavaScript может содержать любую комбинацию данных различного типа. В нашем примере массив содержит данные трех типов: `number`, `string` и `boolean`.

Использование литеральных массивов

Использование литеральной нотации позволяет совместить создание массива и заполнение его значениями в одной инструкции, как показано в листинге 4.28.

Листинг 4.28. Использование литеральной нотации для массивов

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

      var myArray = [100, "Адам", true];

    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом примере создается массив `myArray`, который заполняется значениями, указанными справа от операции присваивания внутри квадратных скобок (`[` и `]`).

Считывание и изменение содержимого массива

Чтобы обратиться к любому элементу массива, следует указать нужный индекс, записав его внутри квадратных скобок после имени массива, как показано в листинге 4.29.

Листинг 4.29. Считывание значений элементов массива с использованием индексов

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myArray = [100, "Адам", true];
        console.log("Индекс 0: " + myArray[0]);
    </script>
</head>
<body>
    Это простой пример
```

```
</body>
</html>
```

Чтобы изменить значение любого элемента массива JavaScript, достаточно присвоить ему новое значение, указав соответствующий индекс. Пример изменения содержимого массива приведен в листинге 4.30.

Листинг 4.30. Изменение содержимого массива

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myArray = [100, "Адам", true];
        myArray[0] = "Среда";
        console.log("Индекс 0: " + myArray[0]);
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этом примере элементу массива `myArray[0]`, который ранее содержал числовое значение, присваивается строковое значение.

Перечисление содержимого массива

Для перечисления (циклического перебора) элементов массива используется цикл `for`. В листинге 4.31 показано, как применить цикл для отображения содержимого простого массива.

Листинг 4.31. Перечисление содержимого массива

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myArray = [100, "Адам", true];
        for (var i = 0; i < myArray.length; i++) {
            console.log("Индекс " + i + ": " + myArray[i]);
        }
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Циклы в JavaScript работают точно так же, как и циклы во многих других языках программирования. Число элементов в массиве можно определить, используя свойство `length`. Результат для листинга 4.31 будет выглядеть так.

```
Индекс 0: 100 Индекс 1: Адам Индекс 2: true
```

Использование встроенных методов объекта `Array`

Объект JavaScript `Array` определяет ряд методов, которые можно использовать для работы с массивами. Наиболее употребительные из этих методов приведены в табл. 4.4.

Таблица 4.4. Методы работы с массивами

Метод	*Описание*	*Тип возвращаемого значения*
`concat(`*другой_массив*`)`	Создает новый массив и объединяет в нем содержимое текущего массива и массива, указанного в качестве аргумента. Допускается указание нескольких аргументов	`Array`
`join(`*разделитель*`)`	Объединяет все элементы массива в строку. Аргумент задает символ, используемый в качестве разделителя	`string`
`pop()`	Обрабатывает массив подобно стеку, возвращая последний элемент и удаляя его из массива	`object`
`push(`*элемент*`)`	Обрабатывает массив подобно стеку, добавляя указанный элемент в конец массива	`void`
`reverse()`	Меняет порядок элемента в массиве на обратный	`Array`
`shift()`	Работает подобно методу `pop`, но воздействует на первый элемент массива	`object`
`slice(`*начало*`,`*конец*`)`	Возвращает подмассив	`Array`
`sort()`	Сортирует элементы массива	`Array`
`unshift(`*элемент*`)`	Работает подобно методу `push`, но вставляет новый элемент в начало массива	`void`

Обработка ошибок

Для обработки ошибок в JavaScript используется блок `try...catch`. В целом при работе с данной книгой мы оставим проблему ошибок в стороне, поскольку наша цель — объяснение возможностей jQuery, а не обучение программированию. Пример использования блока `try...catch` приведен в листинге 4.32.

Листинг 4.32. Обработка ошибок

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        try {
            var myArray;
            for (var i = 0; i < myArray.length; i++) {
                console.log("Индекс " + i + ": " + myArray[i]);
            }
        } catch (e) {
            console.log("Ошибка: " + e);
```

```
        }
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В этот сценарий намеренно введена одна довольно распространенная ошибка: здесь делается попытка использовать переменную еще до того, как ей было присвоено какое-либо значение. Фрагмент кода, относительно которого имеются опасения, что он может служить источником ошибок, помещается в блок `try`. Если никакие проблемы не возникают, то инструкции выполняются обычным образом, и блок `catch` игнорируется.

В случае возникновения ошибки выполнение инструкций немедленно прекращается, и управление передается блоку `catch`. Информация о возникшей ошибке содержится в объекте `Error`, который передается блоку `catch`. Свойства объекта `Error` перечислены в табл. 4.5.

Таблица 4.5. Свойства объекта `Error`

Свойство	*Описание*	*Тип возвращаемого значения*
`message`	Описание состояния сбоя	`string`
`name`	Имя ошибки. По умолчанию таковым является `Error`	`string`
`number`	Номер ошибки данного типа, если таковой существует	`number`

Блок `catch` предоставляет возможность выполнить необходимые восстановительные действия для корректного выхода из состояния сбоя. Если имеются инструкции, которые необходимо выполнить независимо от того, произошла ли ошибка, их можно поместить в необязательный блок `finally`, как показано в листинге 4.33.

Листинг 4.33. Использование блока `finally`

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        try {
            var myArray;
            for (var i = 0; i < myArray.length; i++) {
                console.log("Индекс " + i + ": " + myArray[i]);
            }
        } catch (e) {
            Console.log("Ошибка: " + e);
        } finally {
            console.log("Эти инструкции всегда выполняются");
        }
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Значения undefined и null

В JavaScript есть два специальных значения, обращение с которыми требует особого внимания. Значение undefined (т.е. *не определено*) возвращается при считывании переменной, которой еще не было присвоено значение, или значения несуществующего свойства объекта. Пример использования значения undefined в JavaScript приведен в листинге 4.34.

Листинг 4.34. Специальное значение undefined

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        var myData = {
            name: "Адам",
            weather: "солнечный",
        };
        console.log("Свойство: " + myData.doesntexist);
    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Результат для листинга 4.34 будет выглядеть так.

```
Свойство: undefined
```

В JavaScript имеется еще одна специальная переменная — null. Значение null имеет несколько иной по сравнению со значением undefined смысл. Значение undefined возвращается в тех случаях, когда до обращения к элементу данных значение ему вообще не присваивалось, тогда как значение null вы используете в тех случаях, когда хотите указать, что элементу данных значение было присвоено, однако оно не относится ни к одному из допустимых типов данных: object, string, number или boolean. Сценарий, помогающий прояснить эту ситуацию, приведен в листинге 4.35.

Листинг 4.35. Использование значений undefined и null

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var myData = {
            name: "Адам",
        };

        console.log("Переменная: " + myData.weather);
        console.log("Свойство: " + ("weather" in myData));
```

```
        myData.weather = "солнечный";
        console.log("Переменная: " + myData.weather);
        console.log("Свойство: " + ("weather" in myData));

        myData.weather = null;
        console.log("Переменная: " + myData.weather);
        console.log("Свойство: " + ("weather" in myData));

    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Здесь мы создаем объект, а затем пытаемся получить значение свойства `weather`, которое не определено.

```
console.log("Переменная: " + myData.weather);
console.log("Свойство: " + ("weather" in myData));
```

У данного объекта отсутствует свойство `weather`, поэтому при обращении к свойству `myData.weather` возвращается значение `undefined`, а при использовании ключевого слова `in` для определения того, содержит ли объект данное свойство, возвращается `false`. Результат работы этих двух инструкций будет выглядеть так.

```
Переменная: undefined
Свойство: false
```

Далее мы присваиваем свойству определенное значение и вновь проверяем, имеет ли объект это свойство. Как и следовало ожидать, проверка показывает, что данное свойство действительно определено в объекте и ему присвоено значение `солнечный`.

```
Переменная: солнечный
свойство: true
```

После этого мы устанавливаем значение свойства равным `null`.

```
MyData.weather = null;
```

Результат этого присваивания оказывается весьма специфическим. Свойство по-прежнему определено в объекте, но мы указали, что с ним не связано никакое значение. После выполнения очередной проверки мы получим следующий результат.

```
Переменная: null
Свойство: true
```

Указанные отличия очень важно учитывать при сопоставлении значений `undefined` и `null`, поскольку `null` — это тип `object`, тогда как `undefined` — это независимый специальный тип данных.

Проверка того, что переменная или свойство имеет значение `null` или `undefined`

Если вы хотите проверить, имеет ли свойство значение `null` или `undefined` (и при этом вам безразлично, какое именно из них было ему присвоено), можете

воспользоваться инструкцией `if` и операцией отрицания (`!`), как показано в листинге 4.36.

Листинг 4.36. Проверка того, имеет ли свойство значение `null` или `undefined`

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">

        var myData = {
            name: "Адам",
            city: null
        };

        if (!myData.name) {
            console.log("name РАВНО null или undefined");
        } else {
            console.log("name НЕ РАВНО null или undefined");
        }

        if (!myData.city) {
            console.log("city РАВНО null или undefined");
        } else {
            console.log("city НЕ РАВНО null или undefined");
        }

    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

В основе этого приема лежит осуществляемое JavaScript приведение типов, так что проверяемые значения обрабатываются как булевы. Если переменная (или свойство) равна `null` или `undefined`, то в результате ее приведения булево значение будет равно `false`.

Различия между `null` и `undefined`

При сравнении этих двух значений у вас имеется возможность выбора. Если вы хотите интерпретировать значения `undefined` и `null` как эквивалентные, используйте логический оператор равенства (==) и полагайтесь на то, что JavaScript выполнит приведение типов. В этом случае значение `undefined` будет считаться равным значению `null`. Если же вы хотите проводить различие между `null` и `undefined`, используйте оператор тождественности (===). Пример использования сравнений обоих типов приведен в листинге 4.37.

Листинг 4.37. Сравнение значений `null` и `undefined` для проверки их равенства или тождественности

```
<!DOCTYPE HTML>
<html>
<head>
```

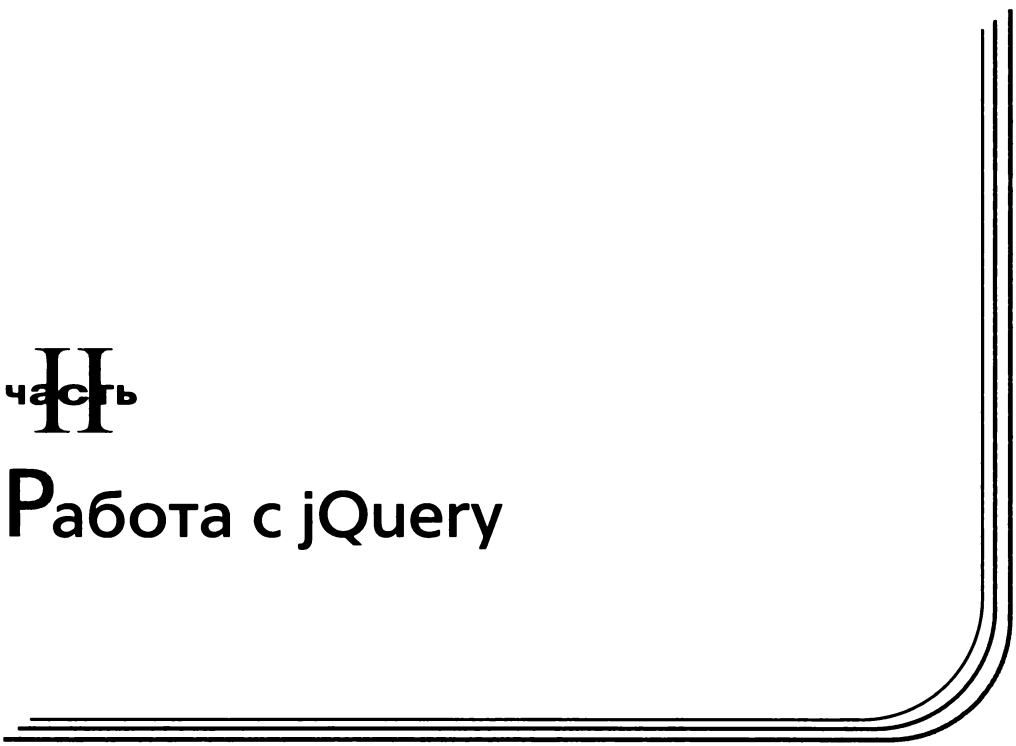
```
    <title>Пример</title>
    <script type="text/javascript">

        var firstVal = null;
        var secondVal;

        var equality = firstVal == secondVal;
        var identity = firstVal === secondVal;

        console.log("Равенство: " + equality);
        console.log("Тождественность: " + identity);

    </script>
</head>
<body>
    Это простой пример
</body>
</html>
```

Результат для этого листинга будет таким.

```
Равенство: true
Тождественность: false
```

Резюме

В этой главе были продемонстрированы основные возможности JavaScript, которые используются в остальных главах книги. Понимание основ JavaScript является залогом эффективного использования jQuery, о чем говорится в последующих главах.



Часть II

Работа с jQuery

ГЛАВА 5

Основы jQuery

В этой главе мы приступаем к работе со сценариями jQuery. Ваш первый сценарий будет не очень сложным, но позволит продемонстрировать основные возможности библиотеки jQuery. Вы научитесь осуществлять отбор нужных элементов в документе, узнаете, что собой представляет выбранный набор и как jQuery взаимодействует со встроенным интерфейсом прикладного программирования DOM-модели (DOM API), являющимся частью спецификации HTML. Перечень тем, рассматриваемых в данной главе, приведен в табл. 5.1.

Таблица 5.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Включение библиотеки jQuery в HTML-документ	Импортируйте библиотеку jQuery с помощью элемента `link`, подключившись либо к своему серверу, либо к CDN. Добавьте элемент `script`, в который будет помещен ваш сценарий jQuery	1, 2
Выбор элементов в документе	Передайте CSS-селектор в функцию `$()` или функцию `jQuery()`	3, 10
Переименование функции `$()`	Используйте метод `noConflict()`	4, 5
Отсрочка выполнения сценария jQuery до полной загрузки документа	Зарегистрируйте обработчик события `ready` для глобальной переменной `document` или передайте функцию в функцию `$()`	6–8
Управление моментом срабатывания события `ready`	Используйте метод `holdReady()`	9
Ограничение области выбора элементов частью документа	Передайте контекст в функцию `$()`	11, 12
Определение селектора, который использовался для создания объекта `jQuery`	Получите значение свойства `selector`	13
Определение контекста, который использовался для создания объекта `jQuery`	Получите значение свойства `context`	14
Создание объекта `jQuery` из объектов `HTMLElement`	Передайте объекты в функцию `$()` в качестве аргументов	15
Перечисление содержимого объекта `jQuery`	Обработайте объект `jQuery` как массив или воспользуйтесь методом `each()`	16, 17
Нахождение определенного элемента в объекте `jQuery`	Используйте методы `index()` или `each()`	18–20

Окончание табл. 5.1

Задача	Решение	Листинг
Применение операции одновременно к нескольким элементам в документе	Вызовите соответствующий метод jQuery для объекта `jQuery`	21
Применение нескольких операций к объекту `jQuery`	Объедините вызовы методов в цепочку	22, 23
Обработка события	Используйте один из методов обработки событий jQuery	24

Установка библиотеки jQuery

Первое, что нужно сделать, приступая к работе с jQuery, — это добавить библиотеку в документ, с которым вы хотите работать. В листинге 5.1 воспроизведен пример документа сайта цветочного магазина, представленный ранее в главе 2.

Листинг 5.1. Пример документа сайта цветочного магазина

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
```

```
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Чтобы ваше внимание было постоянно сосредоточено на основном содержимом документа, стили CSS вынесены во внешнюю таблицу стилей, сохраненную в файле `styles.css`, которая обсуждалась в главе 3. В этом примере библиотека jQuery добавляется в документ с помощью следующего кода:

```
<script src="jquery-1.7.js" type="text/javascript"></script>
```

На веб-сайте `jquery.com` для загрузки предлагаются два файла. Первый из них, `jQuery-1.7.js`, содержит версию библиотеки, которую обычно используют при разработке веб-сайтов или приложений. В этом файле находится несжатый JavaScript-код, объем которого составляет примерно 230 Кбайт. Можете свободно открыть этот файл и изучить его содержимое, чтобы посмотреть, как jQuery реализует свои возможности, что позволит вам без труда развернуть стек вызовов в случае возникновения проблем с кодом.

Примечание. На момент написания данной книги последней версией jQuery была 1.7. Библиотека jQuery активно разрабатывается, и можно не сомневаться, что к тому времени, когда вы будете читать эти строки, успеет выйти новая версия. Но и в этом случае демонстрируемые в книге методики по-прежнему будут работать.

Второй файл, `jquery.1.7.min.js`, используется при развертывании сайта или веб-приложения для пользователей. Он содержит тот же JavaScript-код, что и первый файл, однако сделан *более компактным* за счет удаления из него всех пробелов и использования односимвольных имен переменных вместо содержательных имен большей длины для экономии места. Минимизированный сценарий библиотеки почти непригоден для чтения кода в целях отладки, но его размер намного меньше и составляет всего лишь 31 Кбайт. Если обслуживается множество веб-страниц, зависящих от библиотеки jQuery, то эта разница может обеспечить значительное снижение трафика, необходимого для загрузки jQuery (а значит, и соответствующих накладных расходов).

Загрузка jQuery с использованием CDN

Вместо того чтобы хранить библиотеку jQuery на своем сервере, можете воспользоваться одной из публично доступных сетей дистрибуции контента (Content Distribution Network — CDN), в которых хранится jQuery. CDN — это географически распределенная серверная сеть, обеспечивающая доставку файлов

конечному пользователю с ближайшего сервера. Существуют две причины, по которым имеет смысл использовать CDN. Во-первых, это ускоряет доставку файла библиотеки jQuery конечному пользователю, поскольку файл загружается с сервера, который расположен ближе всего по отношению к нему, а не с ваших серверов. Во многих случаях сам файл может вообще не потребоваться. Библиотека jQuery настолько популярна, что существует большая вероятность того, что браузер пользователя ранее уже кэшировал ее файл из другого приложения, которое также использует jQuery. Во-вторых, при таком способе доставки библиотеки jQuery пользователю затраты на это вашего ценнейшего ресурса — полосы пропускания — полностью исключаются. Для сайтов с интенсивным трафиком это может дать значительную экономию средств.

Используя CDN, вы должны быть твердо уверены в надежности ее оператора. Вы должны быть уверены в том, что пользователь получит именно те файлы, на которые рассчитывает, и что служба будет оставаться всегда доступной. Google и Microsoft также предоставляют бесплатные услуги CDN по доставке библиотеки jQuery (равно как и других популярных библиотек JavaScript). Обе компании имеют богатейший опыт бесперебойного предоставления услуг, и от них вряд ли можно ожидать самовольного внесения каких-либо изменений в библиотеку jQuery. Подробнее о службе Microsoft можно узнать по такому адресу:

`http://www.asp.net/ajaxlibrary/cdn.ashx`

Ниже приведен адрес, по которому можно получить информацию о службе Google:

`http://code.google.com/apis/libraries/devguide.html`

Подход, основанный на использовании CDN, невыгоден в случае приложений, доставляемых пользователям по локальной сети, поскольку он приведет к тому, что все серверы будут вынуждены обращаться в Интернет для получения библиотеки jQuery, а не к локальному серверу, который, как правило, находится ближе и в состоянии обеспечить более быструю доставку файлов при одновременной экономии полосы пропускания.

Первый сценарий jQuery

Добавив библиотеку jQuery в документ, можно приступать к созданию сценариев, использующих функциональность jQuery. Пример простого сценария, в котором демонстрируются некоторые базовые возможности библиотеки jQuery, представлен в листинге 5.2.

Листинг 5.2. Первый сценарий jQuery

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function () {
            $("img:odd").mouseenter(function(e) {
                $(this).css("opacity", 0.5);
            }).mouseout(function(e) {
                $(this).css("opacity", 1.0);
            });
        });
    </script>
</head>
<body>
```

```
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Этот сценарий совсем небольшой, однако позволяет познакомиться с некоторыми из наиболее важных возможностей и особенностей jQuery. В данной главе мы проанализируем работу этого сценария строка за строкой, хотя для полного понимания всех охватываемых им областей функциональности вам понадобится прочитать все оставшиеся главы данной части. Прежде всего взгляните на рис. 5.1, на котором показан вид готовой веб-страницы в окне браузера.

Сценарий изменяет непрозрачность (или, что эквивалентно, прозрачность) изображений нарцисса, пиона и подснежника при наведении на них указателя мыши. Следствием этого является незначительное увеличение яркости изображения и его потускнение. При выходе указателя за пределы изображения восстанав-

ливается прежнее значение непрозрачности. На изображения астры, розы и примулы манипуляции мышью никак не влияют.

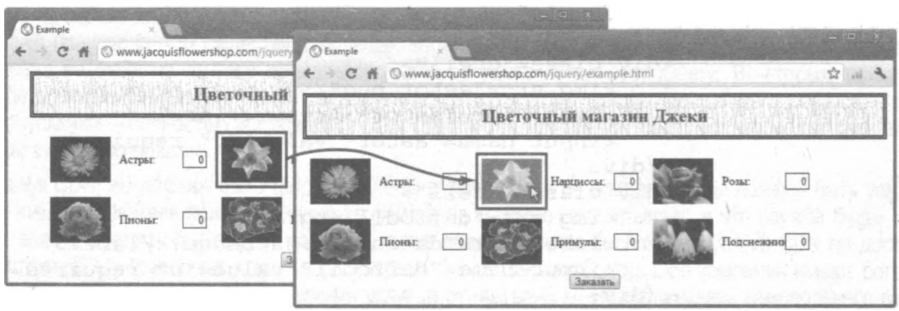


Рис. 5.1. Увеличение прозрачности изображения

Функция $() библиотеки jQuery

Доступ к возможностям jQuery осуществляется с помощью функции библиотеки jQuery `$(...)`, которую мы будем называть просто *функцией $()*. Эта функция служит точкой входа в волшебный мир jQuery, а символ `$` является удобным коротким псевдонимом пространства имен `jQuery`. При желании можете переписать сценарий так, чтобы в нем фигурировало полное имя функции, как показано в листинге 5.3.

Листинг 5.3. Использование полного имени функции `jQuery()` вместо псевдонима

```
...
<script type="text/javascript">
    jQuery(document).ready(function () {
        jQuery("img:odd").mouseenter(function(e) {
            jQuery(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            jQuery(this).css("opacity", 1.0);
        });
    });
</script>
...
```

Данный сценарий обеспечивает ту же функциональность, что и предыдущий. Он требует выполнения большего объема работы по набору текста, но обладает тем преимуществом, что функция `jQuery()` применяется в нем в явном виде.

Библиотека jQuery — не единственная библиотека JavaScript, в которой используется переменная `$`, что может привести к конфликтам имен, если в одном документе используется одновременно несколько библиотек. Чтобы не допустить возникновения проблем такого рода, можно передать контроль над переменной `$` другим библиотекам, вызвав метод `jQuery.noConflict()`, как показано в листинге 5.4.

Листинг 5.4. Отказ от контроля переменной `$` библиотекой `jQuery`

```
...
<script type="text/javascript">
    jQuery.noConflict();
```

```
    jQuery(document).ready(function () {
        jQuery("img:odd").mouseenter(function(e) {
            jQuery(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            jQuery(this).css("opacity", 1.0);
        });
    });
</script>
...
```

Вы даже можете сами определить псевдоним для функции `jQuery()`. Это делается путем присваивания выбранной вами переменной результата вызова метода `noConflict()`, как показано в листинге 5.5.

Листинг 5.5. Использование альтернативного псевдонима

```
...
<script type="text/javascript">
    var jq = jQuery.noConflict();
    jq(document).ready(function () {
        jq("img:odd").mouseenter(function(e) {
            jq(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            jq(this).css("opacity", 1.0);
        });
    });
</script>
...
```

В этом примере для функции `jQuery()` создается новый псевдоним, `jq`, который можно использовать в последующих сценариях.

Примечание. На протяжении всей книги мы будем пользоваться символом `$`, поскольку это общепринятый псевдоним функции `jQuery()` (а также в связи с тем, что никакие другие библиотеки, которые могли бы контролировать эту переменную, нами не используются).

Независимо от выбранного способа обращения к основной функции `jQuery()`, ей передается один и тот же набор аргументов. Возможные варианты вызова этой функции перечислены в табл. 5.2. Последний из них описан в главе 7, тогда как описания остальных вариантов приведены в этой главе.

Таблица 5.2. Варианты вызова основной функции `jQuery()`

Вариант вызова	*Описание*
`$(`*функция*`)`	Позволяет указать функцию, которая должна быть выполнена по завершении построения DOM
`$(`*селектор*`)` `$(`*селектор, контекст*`)`	Осуществляет выбор группы элементов в документе с помощью селектора
`$(HTMLElement)` `$(HTMLElement[])`	Создает объект `jQuery` из объекта или массива объектов `HTMLElement`
`$()`	Создает пустой набор элементов
`$(`*HTML-код*`)`	Создает новые элементы из фрагмента HTML-кода

Ожидание готовности DOM-модели

В главе 2 элемент `script` был помещен в конце документа, чтобы браузер успел создать все объекты в DOM до того, как начнет выполняться код JavaScript. Библиотека jQuery предлагает элегантный способ решения этой проблемы. Соответствующий код представлен в листинге 5.6.

Листинг 5.6. Ожидание завершения построения DOM

```
...
<script type="text/javascript">
    $(document).ready(function () {
        // ...выполняемый код...
    });
</script>
...
```

В этом сценарии мы передаем переменную `document` (которая уже встречалась в главе 1) функции `$()` в качестве аргумента и вызываем метод `ready()`, передавая ему функцию, которую хотим выполнить после окончания загрузки и готовности DOM к работе. Можете поместить этот элемент `script` в любое место документа, будучи уверенным в том, что jQuery не допустит преждевременного выполнения функции.

Примечание. Передача методу `ready()` функции `function()` в качестве аргумента сопровождается созданием обработчика события `ready`. Более подробно события обсуждаются в главе 9. А пока что вам достаточно знать лишь то, что функция `function()`, передаваемая методу `ready()`, будет вызвана лишь после загрузки документа, но не раньше, чем завершится построение DOM.

Последствия пропуска ключевого слова `function` при вызове метода `ready()`

Часто совершают ошибку, опуская в этой магической формуле ключевое слово `function`, определяющее следующий за ним блок инструкций как анонимную функцию, и передавая методу `ready()` простую последовательность инструкций JavaScript. Это не сработает. Инструкции будут выполнены браузером сразу же после их синтаксического разбора, а не после того, как DOM-дерево будет готово к использованию. В этом позволяет убедиться листинг 5.7.

Листинг 5.7. Последствия пропуска ключевого слова `function` при вызове обработчика события `ready`

```
...
    <script type="text/javascript">

        function countImgElements() {
            return $('img').length;
        }

        $(document).ready(function() {
            console.log("Вызвана функция ready.
```

```
            Счетчик IMG: " + countImgElements());
        });

        $(document).ready(
            console.log("Вызвана инструкция ready.
              Счетчик IMG: " + countImgElements())
        );
    </script>
...
```

Здесь метод `ready()` вызывается дважды: первый раз — с использованием ключевого слова `function`, а второй — с передачей обычной инструкции JavaScript в качестве аргумента. В обоих случаях вызывается функция `countImgElements()`, возвращающая общее количество элементов `img` в DOM. (Не пытайтесь сейчас анализировать работу этой функции. О свойстве `length` мы позднее поговорим более подробно.) Загрузив документ, вы получите в окне консоли следующий результат.

```
Вызвана инструкция ready. Счетчик IMG: 0
Вызвана функция ready. Счетчик IMG: 6
```

Как видите, выполнение инструкции без ключевого слова `function` происходит при загрузке документа еще до того, как браузер обнаружит в нем элементы `img` и создаст соответствующие DOM-объекты.

Использование альтернативной нотации

При желании можете передать свою функцию в качестве параметра непосредственно `$`-функции jQuery. При таком способе записи вызова результат будет тем же, что и в случае вызова `$(document).ready()`. Описанный подход используется в листинге 5.8.

Листинг 5.8. Отсрочка выполнения функции до момента готовности DOM

```
<script type="text/javascript">
    $(function() {
        $("img:odd").mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        })
    });
</script>
```

Задержка срабатывания события `ready`

Используя метод `holdReady()`, можно управлять моментом срабатывания события `ready`. Это может пригодиться в тех случаях, когда вы хотите использовать динамическую загрузку внешних ресурсов (эффективный, но пока что редко применяемый прием). Метод `holdReady()` следует вызывать дважды: до срабатывания события `ready` и когда DOM достигнет состояния готовности. Пример использования этой методики приведен в листинге 5.9.

Листинг 5.9. Использование метода `holdReady()`

```
<script type="text/javascript">
    $.holdReady(true);
    $(document).ready(function() {
        console.log("Сработало событие ready");
        $("img:odd").mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        })
    });
    setTimeout(function() {
        console.log("Отмена задержки");
        $.holdReady(false);
    }, 5000);
</script>
...
```

Первой инструкцией в этом сценария является вызов метода `holdReady()`. В качестве аргумента ему передается значение `true`, указывающее на необходимость задержки срабатывания события `ready`. Далее мы определяем функцию, которая должна быть выполнена при срабатывании события `ready` (она содержит тот же набор инструкций, который использовался в начале главы для изменения непрозрачности изображений, но оформленный в виде функции).

Наконец, мы используем метод `setTimeout()` для вызова функции по истечении 5 тыс. миллисекунд. Эта функция содержит вызов метода `holdReady()` с аргументом `false`, указывающим jQuery на необходимость освобождения события `ready` для его последующей обработки. Конечный результат состоит в том, что событие `ready` срабатывает с задержкой в 5 секунд. В сценарий включены также отладочные инструкции, которые после загрузки документа в браузер выводят на консоль следующую информацию.

```
Отмена задержки
Сработало событие ready
```

Совет. Метод `holdReady()` можно вызывать многократно, но количество вызовов с аргументом `true` должно совпадать с количеством вызовом с аргументом `false`, прежде чем будет запущено событие `ready`.

Выбор элементов

Одна из самых важных областей применения функциональности jQuery — это выбор элементов DOM. В качестве примера в листинге 5.10 показано, как осуществить выбор *нечетных* элементов `img`.

Чтобы выбрать элементы, вы просто передаете селектор функции `$()`. Библиотека jQuery поддерживает все множество CSS-селекторов, описанных в главе 3, а

Листинг 5.10. Выбор элементов DOM

```
<script type="text/javascript">
    $(document).ready(function() {
        $("img:odd").mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        })
    });
</script>
```

также некоторые дополнительные селекторы, которые обеспечивают удобные возможности детализированного управления процессом выбора элементов. В данном примере используется псевдоселектор `:odd`, который выбирает нечетные элементы, соответствующие основной части селектора (в данном случае это селектор `img`, который выбирает все элементы `img`, о чем уже говорилось в главе 2). В случае использования селектора `:odd` отсчет элементов начинается с нуля, т.е. первый элемент является четным. Поначалу это может сбивать вас с толку. Наиболее полезные селекторы jQuery перечислены в табл. 5.3.

Совет. Вызвав функцию `$()` без указания аргументов (`$()`), можно создать пустой набор элементов. Я упоминаю об этой возможности исключительно ради полноты изложения, поскольку примеры полезного применения этой возможности мне еще не встречались.

Таблица 5.3. Расширенные селекторы jQuery

Селектор	*Описание*
`:animated`	Выбирает все анимируемые в данный момент элементы
`:contains(`*текст*`)`	Выбирает все элементы, содержащие указанный текст
`:eq(`*n*`)`	Выбирает элемент с индексом *n* (индексы отсчитываются от нуля)
`:even`	Выбирает все четные элементы (индексы отсчитываются от единицы)
`:first`	Выбирает первый из подходящих элементов
`:gt(`*n*`)`	Выбирает все элементы, индекс которых превышает *n* (индексы отсчитываются от нуля)
`:has(`*селектор*`)`	Выбирает элементы, которые содержат хотя бы один элемент, соответствующий указанному селектору
`:last`	Выбирает последний из подходящих элементов
`:lt(`*n*`)`	Выбирает все элементы, индекс которых меньше *n* (индексы отсчитываются от нуля)
`:odd`	Выбирает все нечетные элементы (индексы отсчитываются от единицы)
`:text`	Выбирает все текстовые элементы

Я назвал эти селекторы наиболее полезными, поскольку они определяют функциональность, обеспечить которую с помощью селекторов CSS было бы нелегко. Эти селекторы используются точно так же, как и псевдоселекторы CSS. Их можно

использовать независимо — и в этом случае они применяются ко всем элементам DOM, как показано ниже.

```
$(':even')
```

Кроме того, они применяются в сочетании с другими селекторами для ограничения сферы их действия.

```
$('img:even')
```

В jQuery определены также селекторы, позволяющие выбирать элементы в соответствии с их типом (табл. 5.4).

Таблица 5.4. Расширенные селекторы типов, определенные в jQuery

Селектор	*Описание*
`:button`	Выбирает все элементы типа `button`
`:checkbox`	Выбирает все элементы типа `checkbox`
`:file`	Выбирает все элементы типа `file`
`:header`	Выбирает все элементы заголовков
`:hidden`	Выбирает все скрытые элементы
`:image`	Выбирает все элементы изображений
`:input`	Выбирает все элементы `input`
`:parent`	Выбирает все элементы, являющиеся родительскими по отношению к другим элементам
`:password`	Выбирает все элементы, являющиеся паролями
`:radio`	Выбирает все элементы типа `radio`
`:reset`	Выбирает все элементы типа `reset`
`:selected`	Соответствует всем выбранным элементам
`:submit`	Выбирает все элементы типа `submit`
`:visible`	Выбирает все видимые элементы

Сужение области поиска с помощью контекста

По умолчанию jQuery осуществляет поиск элементов в пределах всего DOM-дерева. Область поиска можно сузить, предоставив функции `$()` дополнительный аргумент. Это позволяет ограничить поиск определенным *контекстом*, который используется в качестве отправной точки при нахождении подходящих элементов. Соответствующий пример приведен в листинге 5.11.

Листинг 5.11. Сужение области поиска с помощью контекста

```
<script type="text/javascript">
    $(document).ready(function() {

        $("img:odd", $('.drow')).mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        })
```

```
        });
    </script>
...
```

В этом примере одно множество элементов, выбранных селектором jQuery, используется в качестве контекста для выбора другого множества. Сначала определяется контекст, которому соответствуют все элементы, имеющие класс `drow`. После этого набор выбранных элементов передается селектору `img:odd` в качестве контекста.

Если вы предоставляете контекст, содержащий несколько элементов, то каждый из них используется в качестве отправной точки для поиска. При этом следует учитывать одну тонкость: сначала выбирается совокупность элементов, соответствующих контексту, а уже после этого из них выбираются элементы с помощью основного селектора. В отношении нашего примера это означает, что селектор `img:odd` применяется ко множеству элементов, отобранных с помощью селектора `drow`, откуда следует, что этот набор нечетных элементов не будет совпадать с тем их набором, который был бы найден при выполнении поиска по всему документу. Конечный результат состоит в том, что эффект изменения непрозрачности применяется к нечетным элементам `img` в каждом элементе `div`, принадлежащем классу `drow`. Этим условиям соответствуют изображения нарцисса и примулы. Если опустить контекст, то эффект будет применен к изображениям нарцисса, пиона и подснежника.

Если поиск подходящих элементов необходимо начать с определенного места в документе, то в качестве контекста можно использовать объект `HTMLElement`. Соответствующий пример приведен в листинге 5.12. В следующем разделе будет продемонстрировано, насколько просто осуществляются переходы от объектов jQuery к объектам `HTMLElement` и обратно.

Листинг 5.12. Использование объектов `HTMLElement` в качестве контекста

```
    <script type="text/javascript">
        $(document).ready(function() {
            var elem = document.getElementById("oblock");

            $("img:odd", elem).mouseenter(function(e) {
                $(this).css("opacity", 0.5);
            }).mouseout(function(e) {
                $(this).css("opacity", 1.0);
            })
        });
    </script>
...
```

В этом сценарии поиск нечетных элементов `img` ограничен элементами, являющимися потомками элемента, значением атрибута `id` которого является `oblock`. Конечно, тот же результат можно было получить с помощью CSS-селектора потомков. Преимущество рассматриваемого нами подхода проявляется в ситуациях, когда необходимо сузить область поиска программным путем, не прибегая к конструированию строки селектора. Примером благоприятной для этого ситуации может служить обработка события. Если хотите получить более подробную информацию о событиях (а также узнать о том, каким образом в подобных ситуациях осуществляется доступ к объектам `HTMLElement`), обратитесь к главе 9.

Что собой представляет выбранный набор элементов

Когда jQuery используется для выбора DOM-элементов, функция `$()` возвращает результат в виде объекта `jQuery`, представляющего нуль или более DOM-элементов. Фактически при выполнении любой операции jQuery, которая изменяет один или несколько элементов, ее результатом почти всегда является объект `jQuery`, что составляет важную особенность библиотеки jQuery, к рассмотрению которой мы вскоре вернемся.

Методы и свойства объекта `jQuery` являются основной темой нашего рассмотрения во всех оставшихся главах книги, однако с наиболее важными из них, перечень которых приведен в табл. 5.5, вы познакомитесь уже в этой главе.

Таблица 5.5. Базовые свойства и методы объекта `jQuery`

Свойство/метод	*Описание*	*Тип возвращаемого значения*
`context`	Возвращает набор элементов, используемых в качестве контекста поиска	`HTMLElement`
`each(`*функция*`)`	Выполняет указанную функцию для каждого из выбранных элементов	`jQuery`
`get(`*индекс*`)`	Получает объект `HTMLElement` с указанным индексом	`HTMLElement`
`index(HTMLElement)`	Производит поиск указанного объекта `HTMLElement` среди набора выбранных элементов и возвращает его индекс, если находит его	`number`
`index(jQuery)`	Аналогичен предыдущему методу, но возвращает индекс первого из элементов, содержащихся в указанном объекте `jQuery`	`number`
`index(`*селектор*`)`	Возвращает индекс первого найденного элемента в объекте `jQuery`, вычисляемый относительно элементов соответствующих селектору	`number`
`length`	Возвращает число элементов в объекте `jQuery`	`number`
`selector`	Возвращает селектор	`string`
`size()`	Возвращает количество элементов, содержащихся в объекте `jQuery`	`number`
`toArray()`	Возвращает объекты `HTMLElement`, содержащиеся в объекте `jQuery`, в виде массива	`HTMLElement`

Определение селектора

Свойство `selector` возвращает селектор, который описывает набор выбранных элементов, содержащийся в объекте `jQuery`. Если вы сужаете или расширяете набор выбираемых элементов (о чем пойдет речь в главе 6), то свойство `selector` возвращает селектор, описывающий объединенный набор операций. Использование свойства `selector` продемонстрировано в листинге 5.13.

Листинг 5.13. Использование свойства `selector`

```
    <script type="text/javascript">
        $(document).ready(function() {
            var selector =  $("img:odd").selector
            console.log("Селектор: " + selector);
        });
    </script>
...
```

Этот сценарий выводит на консоль следующий результат.

```
Селектор: img:odd
```

Определение контекста

Свойство `context` предоставляет подробную информацию о контексте, который использовался при создании объекта `jQuery`. Если в качестве контекста использовался единственный объект `HTMLElement`, то он и будет возвращен свойством `context`. Если же в качестве контекста использовалось несколько элементов (как в приведенном ранее примере) или он вообще отсутствовал, то свойство `context` возвратит значение `undefined`. Пример использования этого свойства представлен в листинге 5.14.

Листинг 5.14. Определение контекста для объекта `jQuery`

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            var jq1 = $("img:odd");
            console.log("Без контекста: " + jq1.context.tagName);

            var jq2 = $("img:odd", $('.drow'));
            console.log("Несколько элементов контекста: " +
              jq2.context.tagName);

            var jq3 = $("img:odd",
              document.getElementById("oblock"));
            console.log("Единственный элемент контекста: " +
              jq3.context.tagName);
        });
    </script>
...
```

В этом сценарии демонстрируются три способа выбора элементов: без контекста, с контекстом в виде нескольких элементов и с контекстом в виде одного элемента. В окне консоли будет выведен следующий результат.

```
Без контекста: undefined
Несколько элементов контекста: undefined
Единственный элемент контекста: DIV
```

Работа с DOM-объектами

Библиотека jQuery не подменяет собой DOM-модель, а лишь намного облегчает работу с ней. Объекты HTMLElement (с которыми вы уже познакомились в главе 2) можно использовать, как и прежде, но jQuery упрощает переход от объектов jQuery к объектам DOM и обратно. По моему мнению, та простота, с которой можно переходить от традиционной DOM-модели к объектам jQuery и обратно и которая позволяет поддерживать обратную совместимость со сценариями и библиотеками, не связанными с jQuery, является следствием элегантности построения самой библиотеки jQuery.

Создание объектов jQuery из DOM-объектов

Объекты jQuery можно создавать, передавая объект или массив объектов HTMLElement функции $() в качестве аргумента. Такой способ удобен при работе с JavaScript-кодом, не ориентированным на jQuery, или в ситуациях, когда jQuery открывает доступ к базовым DOM-объектам, например при обработке событий. Соответствующий пример приведен в листинге 5.15.

Листинг 5.15. Создание объектов jQuery из DOM-объектов

```
    <script type="text/javascript">
        $(document).ready(function() {

            var elems = document.getElementsByTagName("img");

            $(elems).mouseenter(function(e) {
               $(this).css("opacity", 0.5);
            }).mouseout(function(e) {
               $(this).css("opacity", 1.0);
            })
        });
    </script>
...
```

В этом примере для выбора элементов img в документе вместо непосредственного использования селекторов jQuery применяется метод document.getElements ByTagName(). Результат работы этого метода (коллекция объектов HTMLElement) передается функции $(), возвращающей обычный объект jQuery, который можно использовать так же, как и в предыдущих примерах.

В данном сценарии попутно демонстрируется создание объекта jQuery из одиночного объекта HTMLElement:

```
$(this).css("opacity", 1.0);
```

При обработке событий средствами jQuery переменная this ссылается на элемент HTMLElement, обрабатывающий событие. Поддержка событий в jQuery описывается в главе 9, а потому мы не будем сейчас углубляться в эту тему (хотя и обсудим далее функции, содержащие соответствующие инструкции).

Работа с объектами jQuery как с массивами

Объект jQuery может рассматриваться и как массив объектов HTMLElement. Это означает, что наряду с развитыми средствами, предлагаемыми библиотекой jQuery, по-прежнему можно использовать объекты DOM. Можете использовать свойство

length или метод size() для определения числа элементов, которые входят в набор выбранных элементов, содержащийся в объекте jQuery, и получать доступ к отдельным DOM-объектам, используя индексную нотацию массивов (скобки [и]).

Совет. Для извлечения объектов HTMLElement из объекта jQuery, рассматриваемого как массив, можно использовать метод toArray(). Лично я стараюсь работать только с объектами jQuery, но иногда, например в случае унаследованного кода, в котором возможности jQuery не используются, удобнее работать непосредственно с DOM-объектами.

Пример перечисления содержимого объекта jQuery с целью доступа к содержащимся в нем элементам HTMLElement приведен в листинге 5.16.

Листинг 5.16. Работа с объектом jQuery как с массивом

```
    <script type="text/javascript">
        $(document).ready(function() {
            var elems = $('img:odd');
            for (var i = 0; i < elems.length; i++) {
                console.log("Элемент: " + elems[i].tagName + " " +
                  elems[i].src);
            }
        });
    </script>
...
```

В этом листинге функция $() используется для выбора нечетных элементов img и их просмотра в цикле с последующим выводом значений свойств tagName и src на консоль. Результат работы сценария выглядит следующим образом.

```
http://www.jacquisflowershop.com/jquery/daffodil.png
http://www.jacquisflowershop.com/jquery/peony.png
http://www.jacquisflowershop.com/jquery/snowdrop.png
```

Итерирование функции по DOM-объектам

Метод each() позволяет определить функцию, которая будет выполнена для каждого из DOM-объектов, содержащихся в объекте jQuery. Соответствующий пример приведен в листинге 5.17.

Листинг 5.17. Использование метода each()

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            $('img:odd').each(function(index, elem) {
                console.log("Элемент: " + elem.tagName + " " +
                  elem.src);
            });
        });
    </script>
```

Библиотека jQuery передает указанной функции два аргумента. Первый из них — это индекс элемента в коллекции, а второй — собственно объект элемента. В данном примере мы выводим на консоль имя дескриптора и значение свойства `src`, получая при этом тот же результат, что и в предыдущем примере.

Определение индекса элемента

Метод `index()` позволяет находить индекс элемента в объекте `jQuery`. В качестве аргумента ему можно передать либо HTML-элемент, либо объект `jQuery`. В последнем случае метод возвращает индекс первого из содержащихся в указанном объекте `jQuery` элемента. Демонстрационный пример приведен в листинге 5.18.

Листинг 5.18. Нахождение индекса HTML-элемента

```
...
    <script type="text/javascript">
        $(document).ready(function() {

            var elems = $('body *');

            // найти индекс с использованием базового DOM API
            var index =
              elems.index(document.getElementById("oblock"));
            console.log("Индекс, найденный с использованием
                DOM-элемента: " + index);

            // найти индекс с использованием другого объекта
            // jQuery
            index = elems.index($('#oblock'));
            console.log("Индекс, найденный с использованием
                объекта jQuery: " + index);
        });
    </script>
...
```

В этом примере сначала выполняется поиск элемента `div` по значению атрибута `id`. Для этого используется метод DOM `getElementById()`, который возвращает объект `HTMLElement`. Затем для нахождения индекса объекта, представляющего элемент `div`, вызывается метод `index()` объекта `jQuery`. Далее этот процесс повторяется с использованием объекта `jQuery`, получаемого посредством функции `$()`, и оба результата выводятся на консоль, как показано ниже.

```
Индекс, найденный с использованием DOM-элемента: 2
Индекс, найденный с использованием объекта jQuery: 2
```

Кроме того, метод `index()` может принимать в качестве аргумента строку. Эта строка интерпретируется как селектор. При этом метод `index()` ведет себя иначе, нежели в предыдущем примере. Соответствующий сценарий приведен в листинге 5.19.

Листинг 5.19. Использование селекторной версии метода `index()`

```
...
    <script type="text/javascript">
        $(document).ready(function() {
```

```
            var imgElems = $('img:odd');
            // найти индекс с использованием селектора
            index = imgElems.index("body *");
            console.log("Индекс, найденный с использованием
                селектора: " + index);

            // выполнить ту же задачу с использованием объекта
            // jQuery
            index = $("body *").index(imgElems);
            console.log("Индекс, найденный с использованием
                объекта jQuery: " + index);

        });
    </script>
...
```

Если в качестве аргумента метода `index()` используется строка, то порядок обработки коллекций элементов изменяется. Сначала jQuery отбирает элементы, соответствующие селектору, а затем определяет в этой совокупности элементов индекс первого из элементов набора, содержащегося в коллекции того объекта `jQuery`, для которого вызывается метод `index()`. Это означает, что оператор

```
index = imgElems.index("body *");
```

эквивалентен следующему оператору:

```
index = $("body *").index(imgElems);
```

По существу, передача методу `index()` строкового аргумента приводит к обмену ролей фигурирующих здесь двух наборов выбранных элементов.

Совет. Использование метода `index()` без аргумента позволяет получить позицию элемента относительно его сестринских элементов. Это может быть полезным в тех случаях, когда jQuery используется для работы с DOM, что обсуждается в главе 7.

Метод `get()` дополняет метод `index()` в том смысле, что позволяет получить объект `HTMLElement`, который занимает в наборе элементов, содержащихся в объекте `jQuery`, позицию, определяемую указанным индексом. Результат получается тем же, что и при использовании индексной нотации массивов, описанной ранее. Соответствующий пример приведен в листинге 5.20.

Листинг 5.20. Получение элемента `HTMLElement` с заданным индексом

```
    <script type="text/javascript">
        $(document).ready(function() {
            var elem = $('img:odd').get(1);
            console.log("Элемент: " + elem.tagName +
              " " + elem.src);
        });
    </script>
...
```

В этом сценарии сначала выбираются нечетные элементы `img`, затем с помощью метода `get()` запрашивается объект `HTMLElement` с индексом 1, и, наконец, на консоль выводятся значения свойств `tagName` и `src`. Результат представлен ниже.

```
http://www.jacquisflowershop.com/jquery/peony.png
```

Изменение нескольких элементов и создание цепочки вызовов методов

Одна из особенностей используемого в jQuery подхода состоит в том, что вызов метода объектом `jQuery` обычно модифицирует все элементы, которые содержатся в данном объекте. Я говорю *обычно*, поскольку некоторые методы выполняют операции, применение которых к нескольким объектам не имеет смысла, и с примерами этого вы еще встретитесь в последующих главах. Пример того, насколько jQuery упрощает работу по сравнению с базовым программным интерфейсом DOM, приведен в листинге 5.21.

Листинг 5.21. Одновременное воздействие на множество элементов

```
    <script type="text/javascript">
        $(document).ready(function() {

            $('label').css("color", "blue");

            var labelElems =
              document.getElementsByTagName("label");
            for (var i = 0; i < labelElems.length; i++) {
                labelElems[i].style.color = "blue";
            }
        });
    </script>
...
```

В этом примере выбираются все элементы `label`, присутствующие в документе, и CSS-свойству `color` каждого из них присваивается значение `blue`. В jQuery это делается всего одной инструкцией, тогда как использование базового программного интерфейса DOM требует несколько больших усилий. Следует признать, что в данном конкретном случае это различие не слишком велико, однако в сложных веб-приложениях оно становится намного более ощутимым. Кроме того, я считаю, что инструкция jQuery более предпочтительна, поскольку ее смысл более очевиден, однако это всего лишь мое личное мнение.

Еще одним удобным свойством объекта `jQuery` является то, что он реализует так называемый *текучий программный интерфейс* (fluent API)[1]. Под этим понимается, что всякий раз, когда вы вызываете метод, изменяющий содержимое объекта, результатом работы этого метода будет другой объект jQuery. Это обстоятельство может казаться довольно несущественным, однако оно позволяет объединять методы в цепочки, как показано в листинге 5.22.

Листинг 5.22. Формирование цепочки вызовов методов jQuery

```
    <script type="text/javascript">
        $(document).ready(function() {
```

[1] Более подробную информацию о "текучих" программных интерфейсах можно найти по адресу `http://ru.wikipedia.org/wiki/Fluent_interface`. — *Примеч. ред.*

```
            $('label').css("color",
              "blue").css("font-size", ".75em");

            var labelElems =
              document.getElementsByTagName("label");
            for (var i = 0; i < labelElems.length; i++) {
                labelElems[i].style.color = "blue";
                labelElems[i].style.fontSize = ".75em";
            }
        });
    </script>
...
```

В этом примере с помощью метода `$()` создается объект `jQuery`, а затем дважды вызывается метод `css()`: сначала — для установки значения свойства `color` созданного объекта, а затем — для установки значения свойства `font-size`. Для сравнения в сценарий включен эквивалентный код, решающий те же задачи, но с использованием базового программного интерфейса DOM. В данном случае объем дополнительной работы оказался незначительным, поскольку уже имелся цикл `for`, осуществляющий перебор выбранных элементов.

Реальные преимущества jQuery проявляются при формировании цепочек методов, вносящих более существенные изменения в набор элементов, которые содержатся в объекте `jQuery`. Соответствующий пример приведен в листинге 5.23.

Листинг 5.23. Более сложный пример цепочки вызовов методов

```
    <script type="text/javascript">
        $(document).ready(function() {

            $('label').css("color",
              "blue").add("input[name!='rose']")
              .filter("[for!='snowdrop']").css("font-size",
              ".75em");

            var elems = document.getElementsByTagName("label");
            for (var i = 0; i < elems.length; i++) {
                elems[i].style.color = "blue";
                if (elems[i].getAttribute("for") != "snowdrop") {
                    elems[i].style.fontSize= ".75em";
                }
            }
            elems = document.getElementsByTagName("input");
            for (var i = 0; i < elems.length; i++) {
                if (elems[i].getAttribute("name") != "rose") {
                    elems[i].style.fontSize= ".75em";
                }
            }
        });
    </script>
...
```

Несмотря на то что в этом примере мы немного забегаем вперед, он позволяет продемонстрировать гибкость jQuery. Проанализируем отдельные звенья цепочки вызовов, чтобы разобраться в том, как она работает. Первый шаг заключается в следующем:

```
$('label').css("color", "blue")
```

Такое вот простое и элегантное начало. Мы выбираем все элементы label, содержащиеся в документе, и устанавливаем для CSS-свойства color каждого из них значение blue. Далее делается следующее:

```
$('label').css("color", "blue").add("input[name!='rose']")
```

Метод add() добавляет в объект jQuery элементы, которые соответствуют указанному селектору. В данном случае выбираются все элементы input, не имеющие атрибута name, значением которого является rose. Эти элементы присоединяются к ранее отобранным элементам, образуя комбинацию элементов label и input. Более полная информация о методе add() приведена в главе 6. Очередной шаг таков:

```
$('label').css("color", "blue").add("input[name!='rose']")
    .filter("[for!='snowdrop']")
```

Метод filter() исключает из объекта jQuery все элементы, которые не удовлетворяют указанному условию. Более подробно этот метод рассматривается в главе 6, а сейчас вам будет достаточно знать лишь то, что он позволяет удалить из объекта jQuery любой элемент, имеющий атрибут for, значением которого является snowdrop. Последний шаг заключается в следующем:

```
$('label').css("color", "blue").add("input[name!='rose']")
    .filter("[for!='snowdrop']").css("font-size", ".75em");
```

Здесь вновь вызывается метод css(), но на этот раз для установки значения свойства font-size равным .075em. Итоговый результат можно описать следующим образом.

1. CSS-свойству color всех элементов label присваивается значение blue.
2. CSS-свойству font-size всех элементов label за исключением того, атрибут for которого имеет значение snowdrop, присваивается значение 0.75em.
3. CSS-свойству font-size всех элементов input, значением атрибута name которых не является rose, присваивается значение 0.75em.

Добиться того же результата с использованием базового DOM API намного сложнее, и при написании данного сценария я столкнулся с некоторыми трудностями. Например, я полагал, что смогу воспользоваться методом document.querySelectrAll(), описанным в главе 2, для выбора элементов input с помощью селектора input[name!='rose'], но оказалось, что фильтры атрибутов такого рода не работают с этим методом. Затем я пытался избежать двойного вызова метода для установки значения свойства font-size путем конкатенации результатов двух вызовов метода getElementsByTagName(), но оказалось, что сделать это также нелегко. Мне не хотелось бы тратить дополнительное время на обсуждение этого вопроса, особенно если учесть, что уже сам факт чтения вами данной книги говорит о вашей решимости активно использовать возможности jQuery, но все же еще раз подчеркну, что добиться обеспечиваемого библиотекой jQuery уровня лаконичности и гибкости, используя лишь базовый программный интерфейс DOM, невозможно.

Обработка событий

Вернувшись к сценарию, которым начинается данная глава, вы увидите, что в содержащуюся в нем в цепочку вызовов включены два метода, выделенные в листинге 5.24 полужирным шрифтом.

Листинг 5.24. Вызов цепочки методов в примере скрипта

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            $("img:odd").mouseenter(function(e) {
                $(this).css("opacity", 0.5);
            }).mouseout(function(e) {
                $(this).css("opacity", 1.0);
            })
        });
    </script>
...
```

Методы `mouseenter()` и `mouseout()` позволяют задать две функции, предназначенные для обработки событий `mouseenter` и `mouseout`, описанных в главе 2. Обработка событий в jQuery описана в главе 9, и здесь я лишь хотел показать, как можно использовать поведение объектов `jQuery` для того, чтобы задать общий метод обработки событий для всех выбранных элементов.

Резюме

В этой главе вы познакомились с первым примером сценария jQuery, который позволил продемонстрировать использование некоторых ключевых возможностей библиотеки jQuery: функции `$`, события `ready` и результирующего объекта `jQuery`. Также было показано, что библиотека jQuery не заменяет собой встроенный программный интерфейс DOM-модели, который является частью спецификации HTML, а дополняет его.

ГЛАВА 6

Работа с набором выбранных элементов

В большинстве случаев работа с jQuery осуществляется в два этапа, следуя одному и тому же шаблону. Первый этап заключается в выборе с помощью функции `$()` начального набора элементов страницы, соответствующих определенному критерию отбора, которые возвращаются в виде содержащего их объекта `jQuery`, а второй — в выполнении над каждым элементом набора одной или нескольких операций, приводящих к созданию конечного набора. В этой главе мы сосредоточим свое внимание на первом из этапов. Будут рассмотрены методы jQuery, позволяющие управлять выбранным набором и корректировать его состав в точном соответствии с конкретными задачами. Также будет показано, как осуществлять навигацию по DOM-узлам с использованием средств jQuery. В обоих случаях обычно исходят из некоего предварительно выбранного набора элементов, воздействуя на который, добиваются того, чтобы в нем остались лишь действительно необходимые элементы. Перечень тем, рассматриваемых в данной главе, приведен в табл. 6.1.

Таблица 6.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Расширение текущего набора элементов	Используйте метод `add()`	1
Сужение текущего набора до одного элемента	Используйте методы `first()`, `last()` и `eq()`	2
Сужение текущего набора до подмножества элементов, индексы которых находятся в пределах указанного диапазона	Используйте метод `slice()`	3
Сужение текущего набора путем использования фильтров	Используйте методы `filter()` и `not()`	4, 5
Сужение текущего набора до подмножества элементов, имеющих определенных потомков	Используйте метод `has()`	6
Преобразование текущего набора в другой набор элементов в виде массива jQuery	Используйте метод `map()`	7
Проверка того, что хотя бы один из элементов текущего набора соответствует указанному условию	Используйте метод `is()`	8
Возврат к предыдущему найденному набору	Используйте метод `end()`	9
Добавление предыдущего найденного набора к текущему набору	Используйте метод `andSelf()`	10
Получение дочерних элементов и потомков элементов текущего набора	Используйте методы `children()` и `find()`	11–13

Окончание табл. 6.1

Задача	Решение	Листинг
Получение родительских элементов для элементов текущего набора	Используйте метод `parent()`	14
Получение предков элементов текущего набора	Используйте метод `parents()`	15
Получение предков элементов текущего набора до тех пор, пока не встретится заданный элемент	Используйте метод `parentsUntil()`	16, 17
Получение ближайшего предка, который соответствует указанному селектору или является заданным элементом	Используйте метод `closest()`	18, 19
Получение ближайшего предка, для которого задан тип позиционирования	Используйте метод `offsetParent()`	20
Получение сестринских элементов для элементов текущего набора	Используйте метод `siblings()`	21, 22
Получение предшествующих или последующих сестринских элементов для элементов текущего набора	Используйте метод `next()`, `prev()`, `nextAll()`, `prevAll()`, `nextUntil()` или `prevUntil()`	23

Расширение набора выбранных элементов

Метод `add()` позволяет добавить в существующий объект `jQuery` дополнительные элементы. Различные варианты вызова этого метода приведены в табл. 6.2.

Таблица 6.2. Варианты вызова метода `add()`

Вариант вызова	Описание
`add(`*селектор*`)` `add(`*селектор, контекст*`)`	Добавляет в текущий набор дополнительные элементы, соответствующие селектору, без учета и с учетом контекста
`add(HTMLElement)` `add(HTMLElement[])`	Добавляет в текущий набор элемент или массив элементов `HTMLElement`
`add(jQuery)`	Добавляет в текущий набор содержимое указанного объекта `jQuery`

Подобно многим методам jQuery, метод `add()` возвращает объект `jQuery`, который можно использовать для вызова других методов, в том числе и для последующих вызовов метода `add()`. Пример использования метода `add()` для расширения ранее найденного набора приведен в листинге 6.1.

Предупреждение. Часто ошибочно полагают, что метод `remove()` является "антиподом" метода `add()` и сокращает выбранный набор. На самом деле метод `remove()` изменяет структуру DOM, о чем говорится в главе 7. Для сокращения набора элементов следует использовать один из методов, о которых говорится в следующем разделе.

Листинг 6.1. Использование метода `add()`

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var labelElems =
                document.getElementsByTagName("label");
            var jq = $('img[src*=daffodil]');

            $('img:even').add('img[src*=primula]').add(jq)
                .add(labelElems).css("border", "thick double
                red");

        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
```

```
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

В этом сценарии для добавления элементов в первоначальный набор используются три подхода: с помощью другого селектора, с помощью объектов `HTMLElement` и с помощью другого объекта `jQuery`. Построив нужный набор объектов, мы вызываем метод `css()`, устанавливая для их свойств `border` значения, обеспечивающие окружение надписей рамками, прорисованными толстыми двойными линиями, как показано на рис. 6.1.

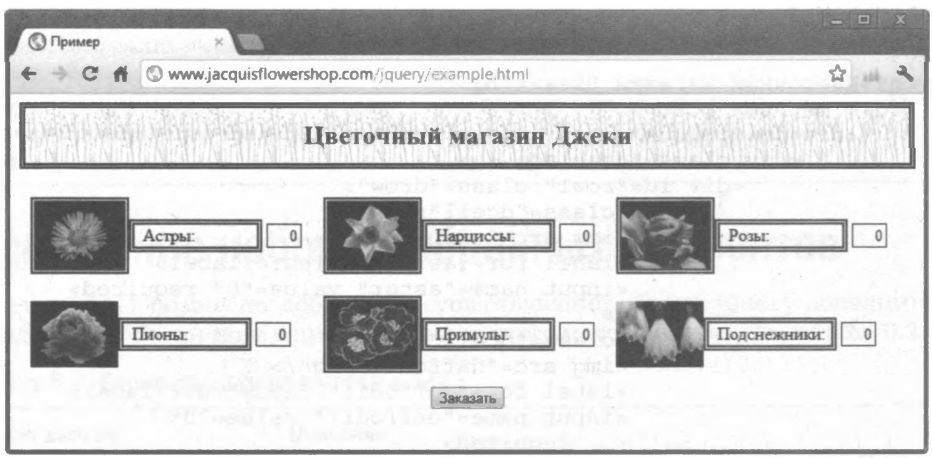


Рис. 6.1. Расширение набора элементов с помощью метода `add()`

Сужение набора выбранных элементов

Существует ряд методов, позволяющих удалять элементы из существующего набора. Их краткое описание приведено в табл. 6.2. Каждый из этих методов возвращает новый объект `jQuery`, содержащий урезанный набор элементов. Объект `jQuery`, для которого вызывается метод, остается неизменным.

Таблица 6.3. Методы фильтрации элементов

Метод	*Описание*
`eq(`*индекс*`)`	Исключает из набора все элементы, кроме элемента с указанным индексом
`filter(`*условие*`)`	Исключает из набора элементы, не соответствующие указанному условию. Подробное описание допустимых типов аргументов приводится далее
`first()`	Исключает из набора все элементы, кроме первого

`has()`
`has(jQuery)`
`has(HTMLElement)`
`has(HTMLElement[])`
	Исключает из набора элементы, у которых отсутствуют потомки, соответствующие указанному селектору или объекту `jQuery`, или потомки, не включающие указанные объекты `HTMLElement`

Окончание табл. 6.3

Метод	Описание
`last()`	Исключает из набора все элементы, кроме последнего
`not(`*условие*`)`	Исключает из набора все элементы, соответствующие указанному условию. Подробное описание различных способов задания условия приводится далее
`slice(`*начало*`,` *конец*`)`	Исключает из набора все элементы, индексы которых выходят за пределы указанного диапазона

Сужение набора до одного элемента

Простейшими методами, с помощью которых можно сократить набор выбранных элементов, являются методы `first()`, `last()` и `eq()`. Эти три метода позволяют выбрать конкретный элемент на основании его позиции в наборе элементов, содержащихся в объекте `jQuery`. Соответствующий пример приведен в листинге 6.2.

Листинг 6.2. Сужение набора на основании позиции элемента

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $('label');

        // выбор первого элемента и воздействие на него
        jq.first().css("border", "thick double red");

        // выбор последнего элемента и воздействие на него
        jq.last().css("border", "thick double green");

        // выбор элемента с указанным индексом и воздействие
        // на него
        jq.eq(2).css("border", "thick double black");
        jq.eq(-2).css("border", "thick double black");

    });
</script>
...
```

Обратите внимание, что метод `eq()` вызывается дважды. Если аргумент имеет положительное значение, отсчет индексов начинается с первого элемента в наборе, содержащемся в объекте `jQuery`. В случае отрицательного значения аргумента индексы отсчитываются в обратном направлении, начиная с последнего элемента. Результат работы сценария приведен на рис. 6.2.

Сужение набора до элементов, индексы которых принадлежат к заданному диапазону

Если необходимо оставить в выбранном наборе лишь элементы, индексы которых принадлежат к заданному диапазону, используется метод `slice()`. Соответствующий пример приведен в листинге 6.3.

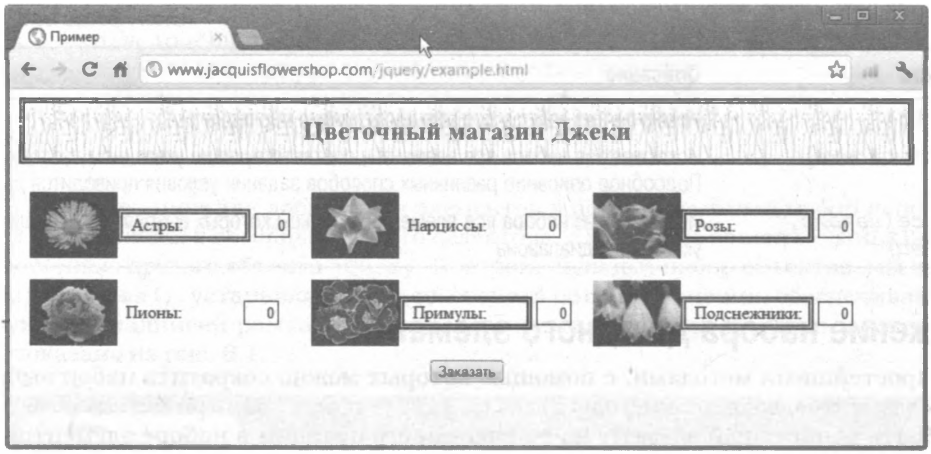


Рис. 6.2. Сужение набора элементов до одного заданного элемента

Листинг 6.3. Использование метода `slice()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $('label');

        jq.slice(0, 2).css("border", "thick double black");
        jq.slice(4).css("border", "thick solid red");

    });
</script>
...
```

В качестве аргументов метод `slice()` принимает значения начального и конечного индексов. Отсчет индексов ведется от нуля, причем элемент, которому соответствует конечный индекс, в результирующий набор не включается. Поэтому аргументам `0` и `2` соответствует выбор первых двух элементов. Если второй аргумент опущен, то выбор элементов продолжается до конца существующего набора. Следовательно, использованию единственного аргумента `4` для набора из шести элементов соответствует выбор последних двух элементов (с индексами `4` и `5`). Результат представлен на рис. 6.3.

Фильтрация элементов

Метод `filter()` позволяет задать условие. Элементы, не удовлетворяющие заданному условию, исключаются из набора. Возможные варианты использования метода `filter()`, соответствующие различным способам задания условия, описаны в табл. 6.4.

Таблица 6.4. Варианты вызова метода `filter()`

Вариант вызова	*Описание*
`filter(`*селектор*`)`	Исключает из набора элементы, которые не соответствуют указанному селектору

Окончание табл. 6.4

Вариант вызова	Описание
`filter(HTMLElement)`	Исключает из набора все элементы, кроме указанного
`flter(jQuery)`	Исключает из набора все элементы, которые не содержатся в указанном объекте `jQuery`
`filter(`*функция*`(`*индекс*`))`	Указанная функция вызывается для каждого элемента набора; из набора исключаются все элементы, для которых функция возвращает значение `false`

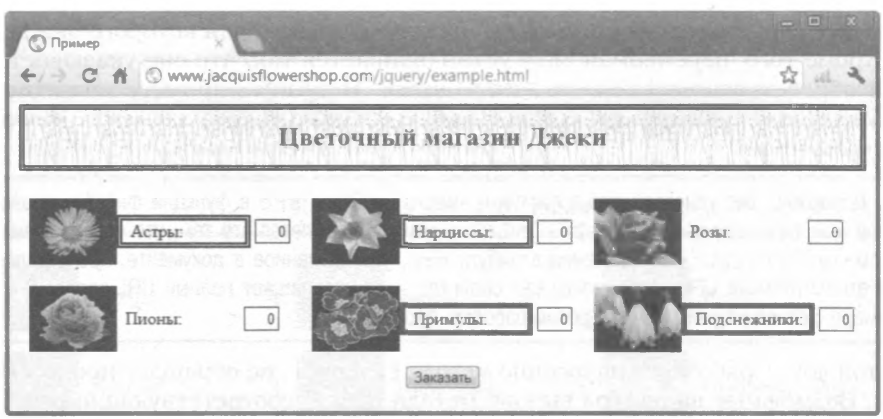


Рис. 6.3. Сужение набора выбранных элементов на основе индексов

Пример использования всех четырех вариантов задания аргументов метода `filter()` приведен в листинге 6.4.

Листинг 6.4. Способы задания фильтра

```
...
<script type="text/javascript">
    $(document).ready(function() {
        // удаление элементов, в значении атрибута src которых
        // содержится буква 's'
        $('img').filter('[src*=s]')
            .css("border", "thick double red");

        // удаление элементов, не содержащих буквы 'p'
        var jq = $('[for*=p]');
        $('label').filter(jq).css("color", "blue");

        // удаление элементов, не являющихся указанным элементом
        var elem = document.getElementsByTagName("label")[1];
        $('label').filter(elem).css("font-size", "1.5em");

        // удаление элементов с использованием функций
        $('img').filter(function(index) {
            return this.getAttribute("src") ==
            "peony.png" || index == 4;
        }).css("border", "thick solid red")
```

```
    });
</script>
...
```

Первые три методики не нуждаются в особых пояснениях. Фильтры в них определяются с помощью селектора, другого объекта `jQuery` и объекта `HTMLElement`. Четвертая методика, основанная на использовании функции, требует дополнительных пояснений. Соответствующие ей строки кода выделены в листинге полужирным шрифтом.

Предоставляемая jQuery функция выполняется по одному разу для каждого элемента набора, содержащегося в объекте `jQuery`. Если эта функция возвращает `true`, то элемент, для которого она была вызвана, остается в наборе. Если же возвращаемым значением является `false`, то элемент удаляется из набора. Функция принимает единственный аргумент — индекс элемента, для которого она вызывается. Кроме того, переменная `this` устанавливается так, что она указывает на текущий обрабатываемый объект `HTMLElement`. В данном примере значение `true` возвращается в тех случаях, когда атрибут `src` или индекс элемента имеют указанные значения.

Совет. Возможно, вас удивляет, что в листинге вместо свойства `src` в функции фильтра вызывается метод `getAttribute()` объекта `HTMLElement`. Объясняется это тем, что возвращаемое методом `getAttribute()` значение *атрибута* `src`, установленное в документе, представляет собой относительный URL-адрес, тогда как *свойство* `src` возвращает полный URL-адрес. В данном примере с относительным URL-адресом работать проще.

Метод `not()` работает аналогично методу `filter()`, но обращает процесс фильтрации. Возможные варианты вызова метода `not()`, соответствующие различным способам задания условия, приведены в табл. 6.5.

Таблица 6.5. Варианты вызова метода `not()`

Вариант вызова	*Описание*
`not(`*селектор*`)`	Удаляет из выбранного набора элементы, соответствующие селектору
`not(HTMLElement[])` `not(HTMLElement)`	Удаляет из выбранного набора указанный элемент или элементы
`not(jQuery)`	Удаляет из выбранного набора элементы, которые содержатся в указанном объекте `jQuery`
`not(`*функция*`(`*индекс*`))`	Указанная функция вызывается для каждого элемента набора; из набора исключаются все элементы, для которых функция возвращает значение `true`

Пример использования метода `not()`, построенный на базе предыдущего примера, приведен в листинге 6.5.

Листинг 6.5. Использование функции в методе `not()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').not('[src*=s]')
            .css("border", "thick double red");

        var jq = $('[for*=p]');
```

```
        $('label').not(jq).css("color", "blue");

        var elem = document.getElementsByTagName("label")[1];
        $('label').not(elem).css("font-size", "1.5em");

        $('img').not(function(index) {
            return this.getAttribute("src") ==
                "peony.png" || index == 4;
        }).css("border", "thick solid red")
    });
</script>
...
```

Результат работы этого сценария показан на рис. 6.4. Как и следовало ожидать, он представляет собой прямую противоположность результату из предыдущего примера.

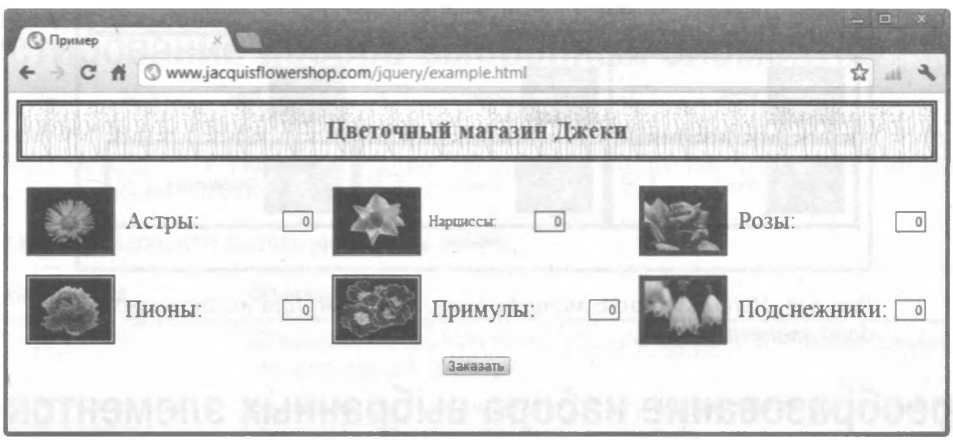


Рис. 6.4. Фильтрация элементов с использованием метода `not()`

Сужение набора до элементов, имеющих определенных потомков

Метод `has()` можно использовать для того, чтобы оставить в наборе выбранных элементов только те из них, у которых есть определенные потомки, указываемые либо с помощью селектора, либо с помощью одного или нескольких объектов `HTMLElement`. Пример использования метода `has()` приведен в листинге 6.6.

Листинг 6.6. Использование метода `has()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('div.dcell').has('img[src*=astor]')
            .css("border", "thick solid red");

        var jq = $('[for*=p]');
        $('div.dcell').has(jq)
            .css("border", "thick solid blue");
```

```
    });
</script>
...
```

В этом сценарии из выбранного набора исключаются элементы, у которых отсутствуют определенные потомки. В первом случае, когда используется селектор, из набора удаляются элементы, не имеющие ни одного потомка `img`, атрибут `src` которого был бы равен `astor`. Во втором случае, когда используется объект `jQuery`, то же самое происходит с элементами, не имеющими ни одного потомка со значением атрибута `for`, содержащим букву `p`. Результат работы сценария показан на рис. 6.5.

Рис. 6.5. Использование метода `has()` для сужения выбранного набора элементов

Преобразование набора выбранных элементов

Метод `map()` обеспечивает гибкий способ использования одного объекта `jQuery` для создания другого. В качестве аргумента методу `map()` передается функция. Эта функция вызывается для каждого из элементов, входящих в исходный объект `jQuery`, а возвращаемые ею объекты `HTMLElement` включаются в результирующий объект `jQuery`.

Пример использования метода `map()` приведен в листинге 6.7.

Листинг 6.7. Пример использования метода `map()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('div.dcell').map(function(index, elem) {
            return elem.getElementsByTagName("img")[0];
        }).css("border", "thick solid red");

        $('div.dcell').map(function(index, elem) {
            return $(elem).children()[1];
        }).css("border", "thick solid blue");

    });
</script>
...
```

В этом сценарии выполняются две операции преобразования. Первая из них, в которой используется программный интерфейс DOM, возвращает первый из элементов `img`, встречающихся в каждом элементе набора. Вторая операция, в которой используются возможности объекта `jQuery`, возвращает первый элемент набора, содержащегося в объекте `jQuery`, который возвращается методом `children()` (этот метод будет подробно обсуждаться далее, однако по его названию вы и сами могли догадаться, что он возвращает дочерние узлы каждого из элементов, содержащихся в объекте `jQuery`).

Совет. Существует возможность возврата лишь одного элемента при каждом вызове функции, передаваемой в качестве аргумента методу `map()`. Если возникает необходимость в преобразовании результирующей коллекции элементов для каждого исходного элемента, можете использовать сочетание методов `each()` и `add()`, о чем говорится в главе 8.

Тестирование набора выбранных элементов

Чтобы выяснить, соответствует ли хотя бы один из выбранных элементов заданному условию, можно использовать метод `is()`. Возможные варианты использования метода `is()`, соответствующие различным типам передаваемых ему аргументов, приведены в табл. 6.6.

Таблица 6.6. Варианты вызова метода `is()`

Вариант вызова	*Описание*
`is(`*селектор*`)`	Возвращает `true`, если объект `jQuery` содержит хотя бы один элемент, соответствующий селектору
`is(HTMLElement[])` `is(HTMLElement)`	Возвращает `true`, если объект `jQuery` содержит указанный элемент или хотя бы один из элементов указанного массива
`is(jQuery)`	Возвращает `true`, если объект `jQuery` содержит хотя бы один из элементов, которые содержатся в объекте, переданном в качестве аргумента
`is(`*функция*`(`*индекс*`))`	Возвращает `true`, если функция возвращает `true` хотя бы один раз

Если в качестве аргумента передается функция, то jQuery вызывает эту функцию для каждого элемента в объекте `jQuery`, передавая ей индекс данного элемента в качестве аргумента и устанавливая переменную `this` так, чтобы она указывала на сам элемент. Пример использования метода `is()` приведен в листинге 6.8.

Примечание. Данный метод возвращает булево (логическое) значение. Как уже отмечалось в главе 5, не все методы возвращают объект `jQuery`.

Листинг 6.8. Использование метода `is()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var isResult = $('img').is(function(index) {
            return this.getAttribute("src") == "rose.png";
        });
```

```
        console.log("Результат: " + isResult);
    });
</script>
...
```

В этом сценарии мы проверяем, содержит ли объект `jQuery` элемент со значением атрибута `src`, равным `rose.png`. На консоль выводится следующий результат.

```
Результат: true
```

Возврат к предыдущему состоянию измененного набора выбранных элементов

В jQuery предусмотрено сохранение состояний выбранного набора элементов при его изменении путем вызова методов цепочки. Из этого обстоятельства можно извлечь выгоду с помощью двух методов, описанных в табл. 6.7.

Таблица 6.7. Методы развертывания стека выборок

Метод	*Описание*
`end()`	Выталкивает текущий выбранный набор из стека и возвращает предыдущий набор
`andSelf()`	Добавляет предыдущий выбранный набор к текущему

Метод `end()` можно использовать для возврата предыдущего выбранного набора, что позволяет выбрать некоторые элементы, расширить или сузить его, выполнить некоторые операции, а затем вернуться к исходному набору, как показано в листинге 6.9.

Листинг 6.9. Использование метода `end()`

```
<script type="text/javascript">
    $(document).ready(function() {

        $('label').first().css("border", "thick solid blue")
            .end().css("font-size", "1.5em");

    });
</script>
```

В этом сценарии мы начинаем с того, что выбираем все элементы `label` в документе. Далее мы сужаем выборку, вызывая метод `first()` (для получения первого подходящего элемента), а затем устанавливаем значение CSS-свойства `body` с помощью метода `css()`.

Следующим вызывается метод `end()` для возврата предыдущего выбранного набора (в результате чего вам вновь предоставляются *все* элементы `label`, а не только первый), а затем опять вызывается метод `css()`, на этот раз для установки значения свойства `font-size`. Результат выполнения сценария показан на рис. 6.6.

Метод `andSelf()` добавляет содержимое предыдущего выбранного набора, находящегося в стеке, к текущему набору. Пример использования метода `andSelf()` приведен в листинге 6.10.

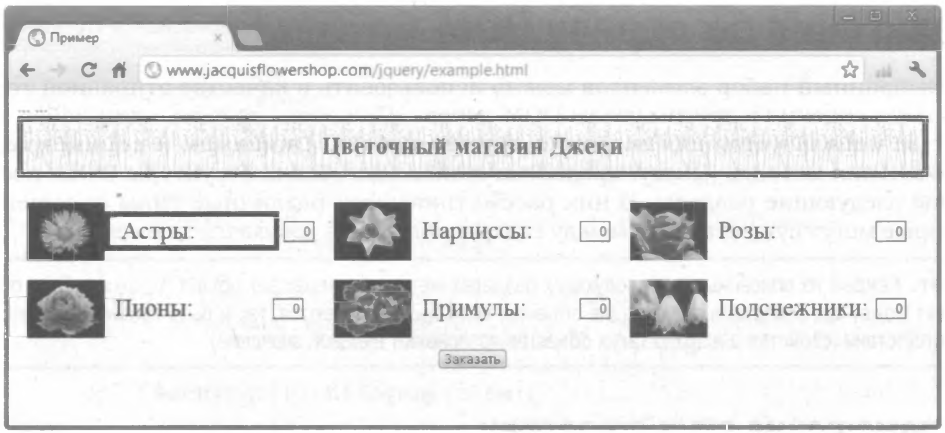


Рис. 6.6. Использование метода `end()`

Листинг 6.10. Использование метода `andSelf`

```
<script type="text/javascript">
    $(document).ready(function() {
        $('div.dcell').children('img').andSelf()
            .css("border", "thick solid blue");
    });
</script>
```

В этом примере сначала выбираются все элементы `div` с классом `dcell`, а затем — все элементы `img`, которые являются их дочерними элементами, для чего используется метод `children()` (более подробная информация об этом методе приводится далее). После этого вызывается метод `andSelf()`, который объединяет предыдущую выборку (элемент `div`) с текущей (элементы `img`) в одном объекте `jQuery`. Наконец для создания рамок вокруг выбранных элементов вызывается метод `css()`. Результат выполнения сценария показан на рис. 6.7.

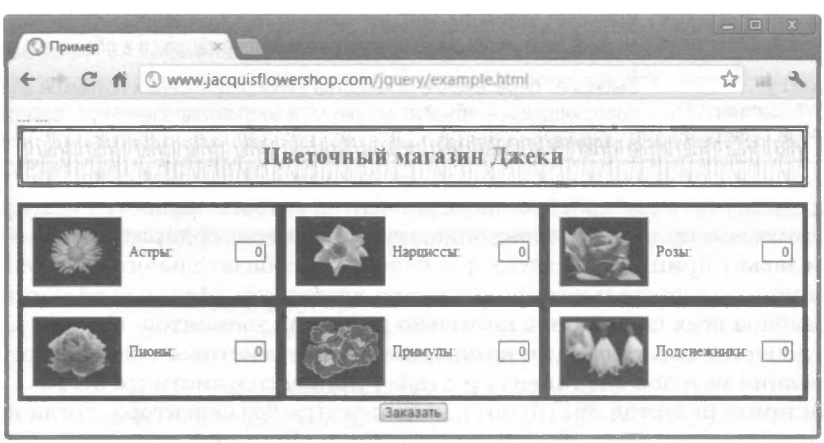


Рис. 6.7. Использование метода `andSelf()`

Навигация по дереву DOM

Выбранный набор элементов можно использовать в качестве отправной точки для перемещения к другим узлам DOM-дерева. При этом, по сути, один набор элементов используется для создания другого набора. Описанию и демонстрации применения методов jQuery, предназначенных для обхода структуры DOM, посвящены следующие разделы. В них рассматриваются различные типы отношений, которые могут существовать между содержащимися в документе элементами.

Совет. Каждый из описанных в последующих разделах методов возвращает объект `jQuery`. Этот объект может как содержать подходящие объекты, если таковые имеются, так и быть пустым в случае их отсутствия (свойство `length` таких объектов возвращает нулевое значение).

Перемещение вниз по дереву

Процесс перемещения вниз по иерархической структуре DOM связан с выбором дочерних элементов (непосредственных потомков), а также всех остальных элементов, являющихся потомками элементов, содержащихся в объекте `jQuery`. Соответствующие методы jQuery и их краткое описание приведены в табл. 6.8.

Таблица 6.8. Методы, используемые для перемещения вниз по иерархической структуре DOM

Метод	*Описание*
`children()`	Выбирает дочерние элементы всех элементов, содержащихся в объекте `jQuery`
`children(`*селектор*`)`	Выбирает все элементы, которые соответствуют указанному селектору и при этом являются непосредственными потомками элементов, содержащихся в объекте `jQuery`
`contents()`	Возвращает дочерние элементы и текстовое содержимое всех элементов, содержащихся в объекте `jQuery`
`find()`	Выбирает потомки элементов, содержащихся в объекте `jQuery`
`find(`*селектор*`)`	Выбирает элементы, которые соответствуют указанному селектору и при этом являются потомками элементов, содержащихся в объекте `jQuery`

`find(jQuery)`
`find(HTMLElement)`
`find(HTMLElement[])`
	Выбирает пересечение множества непосредственных потомков элементов, содержащихся в объекте `jQuery`, и множества элементов, содержащихся в объекте аргумента

Метод `children()` выбирает лишь те элементы, которые являются непосредственными потомками (дочерними элементами) элементов, содержащихся в объекте `jQuery`, и может принимать селектор в качестве необязательного аргумента, обеспечивающего дополнительную фильтрацию элементов. Метод `find()` предназначен для выбора всех потомков, а не только дочерних элементов. Метод `contents()` наряду с дочерними элементами возвращает также текстовое содержимое. Пример использования методов `children()` и `find()` приведен в листинге 6.11.

В этом примере метод `children()` используется без селектора, тогда как метод `find()` используется с одним селектором. Подробная информация о выбранных элементах выводится на консоль вместе с указанием их количества.

Листинг 6.11. Использование методов `children()` и `find()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var childCount = $('div.drow').children()
            .each(function(index, elem) {
            console.log("Дочерний элемент: " + elem.tagName +
                " " + elem.className);
        }).length;
        console.log("Всего имеется " + childCount +
            " дочерних элементов");

        var descCount = $('div.drow').find('img')
            .each(function(index, elem) {
            console.log("Потомок: " + elem.tagName + " " +
                elem.src);
        }).length;
        console.log("Всего имеется " + descCount +
            " элементов-потомков img");

    });
</script>
...
```

```
DIV dcell
DIV dcell
DIV dcell
DIV dcell
DIV dcell
DIV dcell
Всего имеется 6 дочерних элементов

Потомок: IMG http://www.jacquisflowershop.com/jquery/astor.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/rose.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/peony.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/primula.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png
Всего имеется 6 элементов-потомков img
```

Среди приятных особенностей методов `children()` и `find()` можно отметить отсутствие дублирования элементов в выбранном наборе. Это подтверждает пример, приведенный в листинге 6.12.

Листинг 6.12. Отсутствие повторяющихся элементов-потомков в сгенерированном наборе

```
...
<script type="text/javascript">
    $(document).ready(function() {


        $('div.drow').add('div.dcell').find('img')
            .each(function(index, elem) {

            console.log("Элемент: " + elem.tagName +
                " " + elem.src);
```

```
        });
    });
</script>
...
```

В этом примере мы начинаем с того, что создаем объект `jQuery`, который содержит все элементы `div` с классом `drow` и все элементы `div` с классом `dcell`. Ключевым моментом здесь является то, что все элементы, принадлежащие классу `dcell`, одновременно являются элементами класса `drow`. Это означает, что мы имеем дело с двумя перекрывающимися множествами элементов-потомков, и в случае использования метода `find()` с селектором `img` это могло бы привести к дублированию элементов в результирующем наборе, поскольку элементы `img` являются потомками элементов `div` обоих классов. Однако jQuery выручает нас и самостоятельно заботится о том, чтобы среди возвращаемых элементов дублирование отсутствовало, что подтверждается результатами, выводимыми на консоль.

```
Потомок: IMG http://www.jacquisflowershop.com/jquery/astor.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/rose.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/peony.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/primula.png
Потомок: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png
```

Использование метода `find()` для создания пересечения наборов

Методу `find()` можно передать в качестве аргумента объект `jQuery`, объект `HTMLElement` или массив объектов `HTMLElement`. В этом случае метод `find()` будет выбирать элементы, образующие пересечение множества дочерних элементов, содержащихся в исходном объекте `jQuery`, с множеством элементов, содержащихся в объекте аргумента. Соответствующий пример приведен в листинге 6.13.

Листинг 6.13. Использование метода `find()` для создания пересечения выбранных наборов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $('label').filter('[for*=p]').not('[for=peony]');
        $('div.drow').find(jq).css("border", "thick solid blue");

    });
</script>
...
```

Преимущество такого подхода состоит в том, что он обеспечивает высокую избирательность при выборе множества элементов, пересекающегося со множеством дочерних элементов. Сначала мы создаем объект `jQuery`, а затем сужаем множество содержащихся в нем элементов за счет использования методов `filter()` и `not()`. Далее мы передаем этот объект в качестве аргумента методу `find()`, который вызывается другим объектом, содержащим все элементы `div`, принадлежащие классу `drow`. Результирующий набор представляет собой пересечение набора потомков элементов `div.drow` и суженного набора элементов `label`. Результат выполнения сценария показан на рис. 6.8.

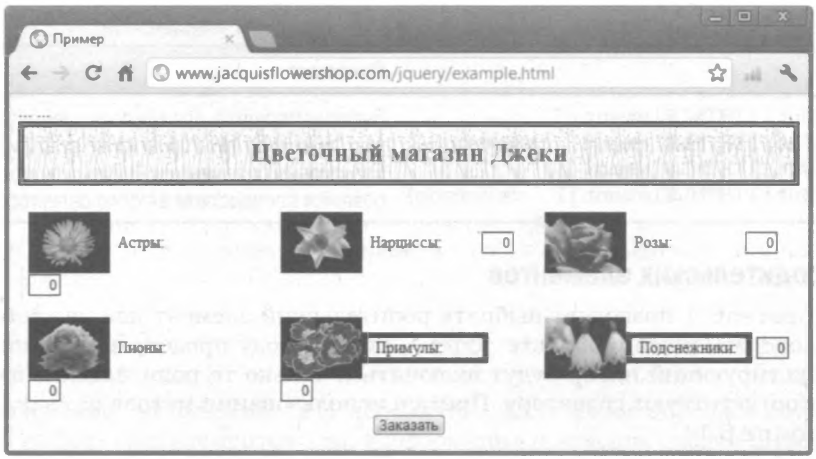


Рис. 6.8. Использование метода `find()` для создания пересечения наборов

Перемещение вверх по дереву

Перемещению вверх по DOM-дереву соответствует поиск родителей и предков элементов, содержащихся в объекте `jQuery`. Методы, используемые для таких перемещений, приведены в табл. 6.9.

Таблица 6.9. Методы, используемые для перемещения вверх по иерархической структуре DOM

Метод	*Описание*
`closest(`*селектор*`)` `closest(`*селектор*`, `*контекст*`)`	Выбор ближайшего предка, соответствующего указанному селектору, для каждого элемента, содержащегося в объекте `jQuery`
`closest(jQuery)` `closest(HTMLElement)`	Выбор ближайшего предка для каждого элемента в объекте `jQuery`, совпадающего с одним из элементов, содержащихся в объекте аргумента
`offsetParent()`	Нахождение ближайшего предка, значением CSS-свойства `position` которого является `fixed`, `absolute` или `relative`
`parent()` `parent(`*селектор*`)`	Выбор непосредственных предков для каждого элемента в объекте `jQuery` с возможностью их фильтрации с помощью селектора
`parents()` `parents(`*селектор*`)`	Выбор предков для каждого элемента в объекте `jQuery` с возможностью их фильтрации с помощью селектора
`parentsUntil(`*селектор*`)` `parentsUntil(`*селектор*`, `*селектор*`)`	Выбор предков для каждого элемента в объекте `jQuery` до тех пор, пока не встретится элемент, соответствующий селектору. Результаты могут фильтроваться посредством второго селектора

Окончание табл. 6.9

Метод	Описание

`parentsUntil(HTMLElement)`
`parentsUntil(HTMLElement, селектор)`
`parentsUntil(HTMLElement[])`
`parentsUntil(HTMLElement[], селектор)`
	Выбирает предков для каждого элемента в объекте `jQuery` до тех пор, пока не встретится один из указанных элементов. Результаты могут фильтроваться посредством второго селектора

Выбор родительских элементов

Метод `parent()` позволяет выбрать родительский элемент для каждого из элементов, содержащихся в объекте `jQuery`. Если методу предоставляется селектор, то в результирующий набор будут включаться только те родительские элементы, которые соответствуют селектору. Пример использования метода `parent()` приведен в листинге 6.14.

Листинг 6.14. Использование метода `parent()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('div.dcell').parent().each(function(index, elem) {
            console.log("Элемент: " + elem.tagName +
                " " + elem.id);
        });

        $('div.dcell').parent('#row1')
            .each(function(index, elem) {
            console.log("Отфильтрованный элемент: " +
                elem.tagName + " " + elem.id);
        });

    });
</script>
...
```

В этом сценарии сначала выбираются все элементы `div`, принадлежащие классу `dcell`, а затем вызывается метод `parent()`, выбирающий все родительские элементы. Здесь также демонстрируется использование метода `parent()` с селектором. Подробная информация о каждом выбранном родительском элементе выводится на консоль с использованием метода `each`.

```
Элемент: DIV row1
Элемент: DIV row2
Отфильтрованный элемент: DIV row1
```

Выбор предков

Метод `parents()` (обратите внимание на последнюю букву `s` в его названии) обеспечивает возможность выбора всех, а не только непосредственных предков (родителей) элементов, содержащихся в объекте `jQuery`. Как и в предыдущем случае, метод может принимать в качестве аргумента селектор для фильтрации результатов. Пример использования метода `parents()` приведен в листинге 6.15.

Листинг 6.15. Использование метода `parents()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img[src*=peony], img[src*=rose]').parents()
            .each(function(index, elem) {
            console.log("Элемент: " + elem.tagName +
                " " + elem.className + " " + elem.id);
        });

    });
    </script>
...
```

В этом примере метод `parents()` используется для выбора предков двух предварительно выбранных элементов `img`. Информация о каждом предке выводится на консоль.

```
Элемент: DIV dcell
Элемент: DIV drow row2
Элемент: DIV dcell
Элемент: DIV drow row1
Элемент: DIV dtable
Элемент: DIV  oblock
Элемент: DIV FORM
Элемент: DIV BODY
Элемент: DIV HTML
```

Метод `parentsUntil()` является еще одной разновидностью методов, предназначенных для выбора предков элементов. Для каждого из элементов, содержащихся в объекте `jQuery`, метод `parentsUntil()` осуществляет перемещение вверх по иерархической структуре DOM, выбирая элементы-предки до тех пор, пока не встретится элемент, соответствующий селектору. Пример использования этого метода приведен в листинге 6.16.

Листинг 6.16. Использование метода `parentsUntil()`

```
...
<script>
    $(document).ready(function() {

        $('img[src*=peony], img[src*=rose]').parentsUntil('form')
            .each(function(index, elem) {
                console.log("Element: " + elem.tagName +
                    " " + elem.className + " " + elem.id);
        });

    });
</script>
...
```

В этом примере процесс выбора предков для каждого элемента продолжается до тех пор, пока не встретится элемент `form`. На консоль выводится следующий результат.

```
Элемент: DIV dcell
Элемент: DIV drow row2
Элемент: DIV dcell
Элемент: DIV drow row1
Элемент: DIV dtable
Элемент: DIV  oblock
```

Обратите внимание, что элементы, соответствующие селектору, исключаются из состава выбираемых предков. В данном случае это означает исключение элемента `form`. Набор предков можно подвергнуть дополнительной фильтрации, предоставив методу `paraentsUntil()` селектор в качестве второго аргумента, как показано в листинге 6.17.

Листинг 6.17. Фильтрация набора элементов, отобранных методом `parentsUntil()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img[src*=peony], img[src*=rose]')
            .parentsUntil('form', ':not(.dcell)')
            .each(function(index, elem) {
            console.log("Элемент: " + elem.tagName +
                " " + elem.className + " " + elem.id);
        });

    });
    </script>
...
```

В этом примере добавлен селектор, который фильтрует элементы, принадлежащие классу `dcell`. На консоль выводится следующий результат.

```
Элемент: DIV drow row2
Элемент: DIV drow row1
Элемент: DIV dtable
Элемент: DIV  oblock
```

Выбор первого подходящего предка

Метод `closest()` позволяет выбирать первого из предков, соответствующих селектору, для каждого элемента в объекте `jQuery`. Пример использования этого метода приведен в листинге 6.18.

Листинг 6.18. Использование метода `closest()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').closest('.drow').each(function(index, elem) {
            console.log("Элемент: " + elem.tagName +
                " " + elem.className + " " + elem.id);
        });
```

```
        var contextElem = document.getElementById("row1");
        $('img').closest('.drow', contextElem)
            .each(function(index, elem) {
            console.log("Контекстный элемент: " + elem.tagName +
                " " + elem.className + " " + elem.id);
        });

    });
</script>
...
```

В этом примере мы сначала выбираем элементы `img` в документе, а затем находим для каждого из них с помощью метода `closest()` ближайшего предка, который принадлежит классу `drow`.

Область выбора предков можно сузить, передав методу `closest()` объект `HTMLElement()` в качестве второго аргумента. Те предки, которые не являются контекстным объектом или его потомками, не включаются в выбранный набор. На консоль выводится следующий результат.

```
Элемент: DIV drow row1
Элемент: DIV drow row2
Контекстный элемент: DIV drow row2
```

Методу `closest()` также можно передать в качестве аргумента объект `jQuery`, объект `HTMLElement` или массив объектов `HTMLElement`. В этом случае метод `closest()` будет продолжать процесс выбора предков для каждого из элементов, содержащихся в исходном объекте `jQuery`, до тех пор пока не встретится объект, переданный в качестве аргумента. Соответствующий пример приведен в листинге 6.19.

Листинг 6.19. Использование метода `closest()` с набором эталонных объектов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $('#row1, #row2, form');

        $('img[src*=rose]').closest(jq)
            .each(function(index, elem) {
            console.log("Контекстный элемент: " + elem.tagName +
                " " + elem.className + " " + elem.id);
        });

    });
</script>
...
```

В этом примере сначала выбирается один из элементов `img` в документе, а затем для нахождения его предков используется метод `closest()`. В качестве аргумента методу передается объект `jQuery`, содержащий элемент `form`, а также элементы с идентификаторами `row1` и `row2`. Из этих элементов jQuery выберет тот, который является ближайшим предком элемента `img`. Иными словами, процесс поиска потомков будет продолжаться до тех пор, пока не встретится один из элементов, указанных в объекте аргумента. На консоль выводится следующий результат.

```
Контекстный элемент: DIV drow row1
```

Метод `offsetParent()` представляет собой вариацию на тему метода `closest()` и предназначен для нахождения первого потомка, значение CSS-свойства `position` которого равно `relative`, `absolute` или `fixed`. Такие элементы называются *позиционированными предками*, и их поиск может оказаться полезным при работе с анимацией (более подробно о поддержке анимации в jQuery говорится в главе 10). Пример использования этого метода приведен в листинге 6.20.

Листинг 6.20. Использование метода `offsetParent()`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        #oblock {position: fixed; top: 120px; left: 50px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('img[src*=astor]').offsetParent()
                .css("background-color", "lightgrey");
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
```

```
    </form>
</body>
</html>
```

В этой усеченной версии документа сначала с помощью CSS устанавливается значение свойства `position` элемента с атрибутом `id` равным `oblock`. Далее в документе выбирается один из элементов `img` и с помощью метода `offsetParent()` находится ближайший позиционированный потомок выбранного элемента. После этого с помощью метода `css()` для свойства `background-color` выбранного элемента устанавливается значение `lighthrey`. Вид результирующей страницы в окне браузера представлен на рис. 6.9.

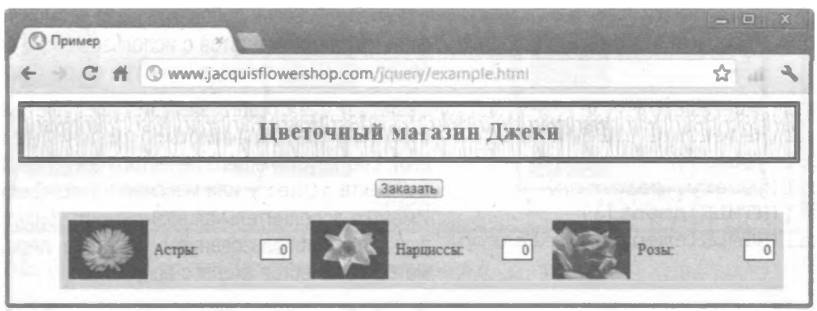


Рис. 6.9. Поиск ближайшего позиционированного предка

Перемещение по дереву в пределах одного иерархического уровня

Последняя разновидность перемещений по DOM-дереву, которую нам осталось рассмотреть, — это перемещение между узлами одного и того же уровня иерархии (между сестринскими узлами). Краткое описание предназначенных для этого методов jQuery приведено в табл. 6.10.

Таблица 6.10. Методы, используемые для перемещения между узлами DOM-дерева в пределах одного иерархического уровня

Метод	*Описание*
`next()` `next(`*селектор*`)`	Выбирает сестринские элементы, непосредственно следующие за каждым из элементов, содержащихся в объекте `jQuery`. Имеется дополнительная возможность фильтрации результатов с использованием селектора
`nextAll()` `nextAll(`*селектор*`)`	Выбирает все последующие сестринские элементы для каждого из элементов, содержащихся в объекте `jQuery`. Имеется дополнительная возможность фильтрации результатов с использованием селектора

`nextUntil(`*селектор*`)`
`nextUntil(`*селектор*`, `*селектор*`)`
`nextUntil(jQuery)`
`nextUntil(jQuery, `*селектор*`)`
`nextUntil(HTMLElement[])`
`nextUntil(HTMLElement[], `*селектор*`)`
	Выбирает для каждого элемента последующие сестринские элементы вплоть до элемента (но не включая его), соответствующего селектору или содержащегося в объекте `jQuery` или массиве `HTMLElement[]`. Имеется дополнительная возможность фильтрации результатов с использованием селектора

Окончание табл. 6.10

Метод	Описание
`prev()` `prev(`*селектор*`)`	Выбирает сестринские элементы, непосредственно предшествующие каждому из элементов, содержащихся в объекте `jQuery`. Имеется дополнительная возможность фильтрации результатов с использованием селектора
`prevAll()` `prevAll(`*селектор*`)`	Выбирает все предшествующие сестринские элементы для каждого из элементов, содержащихся в объекте `jQuery`. Имеется дополнительная возможность фильтрации результатов с использованием селектора, передаваемого методу в качестве второго аргумента

`prevUntil(`*селектор*`)`
`prevUntil(`*селектор*`, `*селектор*`)`
`prevUntil(jQuery)`
`prevUntil(jQuery, `*селектор*`)`
`prevUntil(HTMLElement[])`
`prevUntil(HTMLElement[], `*селектор*`)`
	Выбирает для каждого элемента предшествующие сестринские элементы вплоть до элемента (но не включая его), соответствующего селектору или содержащегося в объекте `jQuery` или массиве `HTMLElement[]`. Имеется дополнительная возможность фильтрации результатов с использованием селектора, передаваемого методу в качестве второго аргумента

`siblings()` `siblings(`*селектор*`)`	Выбирает все сестринские элементы для каждого из элементов, содержащихся в объекте `jQuery`. Имеется дополнительная возможность фильтрации результатов с использованием селектора

Выбор всех сестринских элементов

Метод `siblings()` обеспечивает возможность выбора всех сестринских элементов для всех элементов, содержащихся в объекте `jQuery`. Пример использования этого метода приведен в листинге 6.21.

Листинг 6.21. Использование метода `siblings()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img[src*=astor], img[src*=primula]')
            .parent().siblings()
            .css("border", "thick solid blue");

    });
</script>
...
```

В этом примере мы сначала выбираем два элемента `img`, затем находим их родительские элементы с помощью метода `parent()`, после чего выбираем сестринские элементы последних с помощью метода `siblings()`. При этом выбираются как предшествующие, так и последующие сестринские элементы и для свойства `border` каждого из них устанавливается определенное значение с помощью метода `css()`. Вид результирующей страницы в окне браузера представлен на рис. 6.10. (Метод `parent()` используется здесь для того, чтобы сделать эффект изменения CSS-свойства более заметным.)

Обратите внимание: выбираются лишь сестринские элементы, но не сами элементы. Разумеется, эта ситуация изменится, если один из элементов, содержащихся в объекте `jQuery`, является сестринским по отношению к другому, как показано в листинге 6.22.

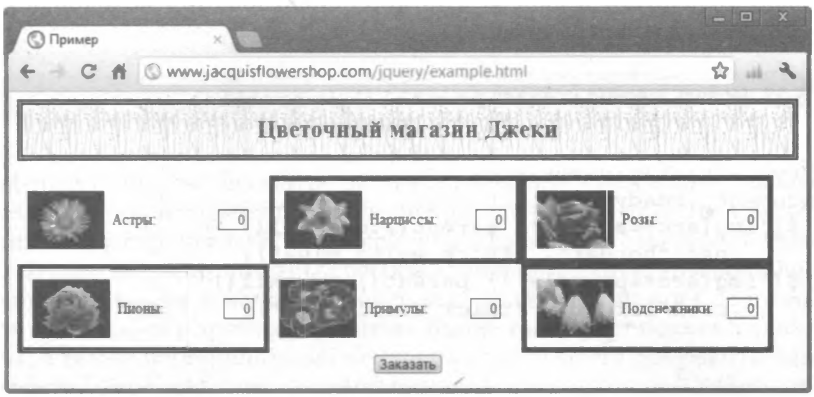


Рис. 6.10. Выбор сестринских элементов

Листинг 6.22. Перекрывающиеся наборы сестринских элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#row1 div.dcell').siblings()
            .css("border", "thick solid blue");
    });
</script>
...
```

В этом сценарии мы начинаем с выбора всех элементов `div`, являющихся дочерними по отношению к элементу `row1`, а затем вызываем метод `siblings()`. Каждый из элементов в выбранном наборе является сестринским в отношении по крайней мере одного из других элементов, как показано на рис. 6.11.

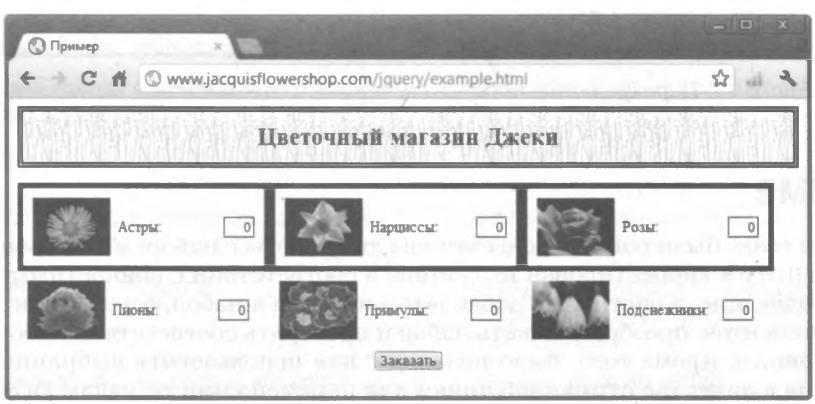


Рис. 6.11. Перекрывание наборов сестринских элементов

Выбор последующих и предшествующих сестринских элементов

Я не собираюсь подробно останавливаться на каждом из методов, предназначенных для выбора предшествующих и последующих сестринских элементов, поскольку все они работают аналогично любому другому методу, обеспечивающему перемещение между узлами DOM-дерева. Пример использования методов `nextAll()` и `prevAll()` приведен в листинге 6.23.

Листинг 6.23. Использование методов `nextAll()` и `prevAll()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img[src*=astor]').parent().nextAll()
            .css("border", "thick solid blue");
        $('img[src*=primula]').parent().prevAll()
            .css("border", "thick double red");
    });
</script>
...
```

В этом сценарии для родительского элемента изображения астры выбираются все последующие сестринские элементы, а для родительского элемента изображения примулы — все предыдущие сестринские элементы. Результат работы сценария представлен на рис. 6.12.

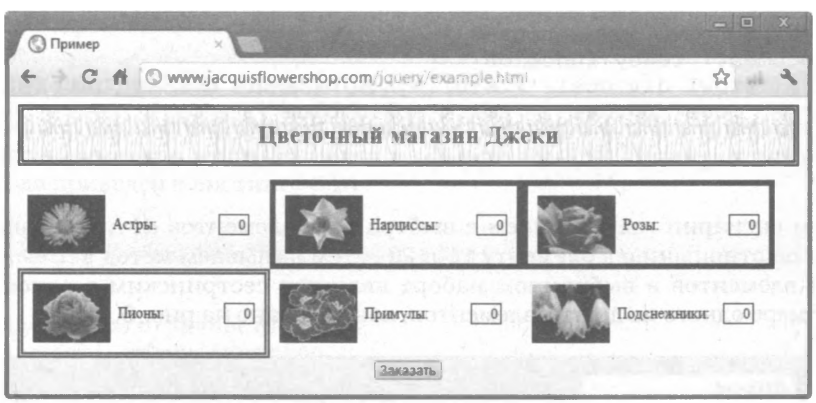


Рис. 6.12. Перекрывание наборов сестринских элементов

Резюме

В этой главе были рассмотрены методы для работы с наборами выбранных элементов jQuery и корректировки их состава в соответствии с конкретными задачами, позволяющие, в частности, добавлять элементы в набор, осуществлять фильтрацию элементов, преобразовывать набор и проверять соответствия набора заданным условиям. Кроме того, было показано, как использовать выбранный набор элементов в качестве отправной точки для перемещении по узлам DOM-дерева с целью создания другого набора элементов документа.

ГЛАВА 7

DOM-манипуляции

В предыдущей главе мы рассмотрели способы выбора элементов. Необычайно широкие возможности открывает использование наборов выбранных элементов для изменения самой структуры HTML-документа, что принято называть *DOM-манипуляциями*. В данной главе описаны способы изменения структуры документа, включая вставку элементов в качестве дочерних, родительских или сестринских элементов по отношению к другим элементам. Кроме того, будет показано, как создавать элементы, а также перемещать элементы из одного места документа в другое или из документа. Перечень тем, рассматриваемых в настоящей главе, приведен в табл. 7.1.

Таблица 7.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Создание элементов	Передайте HTML-фрагмент функции `$`, используя метод `clone()` или DOM API	1–3
Вставка элементов в качестве последних дочерних элементов	Используйте метод `append()`	4
Вставка элементов в качестве первых дочерних элементов	Используйте метод `prepend()`	5, 6
Вставка одних и тех же элементов в различные позиции	Клонируйте элементы перед их вставкой	7, 8
Вставка содержимого объекта `jQuery` в качестве дочернего элемента других элементов	Используйте метод `appendTo()` или `prependTo()`	9
Динамическая вставка дочерних элементов	Передайте функцию методу `append()` или `prepend()`	10
Вставка родительских элементов	Используйте метод `wrap()`	11
Вставка элементов как родительских по отношению к нескольким элементам	Используйте метод `wrapAll()`	12, 13
Окружение элементов на странице заданным содержимым	Используйте метод `wrapInner()`	14
Динамическое окружение элементов заданным содержимым	Передайте функцию методу `wrap()` или `wrapInner()`	15
Вставка сестринских элементов	Используйте метод `after()`, `before()`, `insertAfter()` или `insertBefore()`	16, 17
Динамическая вставка сестринских элементов	Передайте функцию методу `before()` или `after()`	18

Окончание табл. 7.1

Задача	Решение	Листинг
Замена одних элементов другими	Используйте метод `replaceWith()` или `replaceAll()`	19
Динамическая замена элементов	Передайте функцию методу `replaceWith()`	20
Удаление элементов из DOM	Используйте метод `remove()` или `detach()`	21–23
Удаление содержимого элемента	Используйте метод `empty()`	24
Удаление родительских элементов других элементов	Используйте метод `unwrap()`	25

Создание новых элементов

Во многих случаях элементы, которые требуется вставить в DOM, необходимо предварительно создавать (о вставке в DOM существующих элементов будет говориться далее). Рассмотрению способов создания содержимого посвящено несколько следующих разделов.

Совет. Важно понимать, что создание элементов не означает их автоматического добавления в DOM. Вы должны снабдить jQuery явными инструкциями относительно того, куда именно должны быть помещены новые элементы, о чем будет говориться далее.

Создание элементов с использованием функции $()

Одним из способов создания элементов является передача строки, содержащей HTML-фрагмент, функции `$()`, которая выполнит синтаксический анализ строки и создаст соответствующие DOM-объекты. Пример того, как это можно сделать, приведен в листинге 7.1.

Листинг 7.1. Создание элементов с использованием функции $()

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var newElems = $('<div class="dcell">
                <img src="lily.png"/></div>');

            newElems.each(function (index, elem) {
                console.log("Новый элемент: " + elem.tagName +
                    " " + elem.className);
            });

            newElems.children().each(function(index, elem) {
```

```
                console.log("Дочерний элемент: " + elem.tagName +
                    " " + elem.src);
            });
        });
    </script></head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

В этом примере на основе одного HTML-фрагмента создаются два элемента: `div` и `img`. Поскольку мы работаем здесь с HTML-кодом, можно использовать фрагменты, содержащие DOM-структуру. В данном случае у элемента `div` имеется дочерний элемент `img`.

Объект jQuery, возвращаемый функцией $(), содержит лишь элементы верхнего уровня, указанные в HTML-фрагменте. Чтобы продемонстрировать это, в сценарии с помощью функции each() реализован вывод на консоль информации о каждом из элементов, содержащихся в объекте jQuery. Однако дочерние элементы не отбрасываются. К ним можно получить доступ, используя обычные методы, предназначенные для перемещения по структуре DOM-дерева (эти методы описаны в главе 6). С этой целью в сценарии вызывается метод children(), и информация о каждом дочернем элементе также выводится на консоль. Результаты работы сценария, выводимые на консоль, выглядят следующим образом.

```
Новый элемент: DIV dcell
Дочерний элемент: IMG http://www.jackuisflowershop.com/
jquery/lily.png
```

Создание новых элементов путем клонирования существующих

Метод clone() позволяет создавать новые элементы на основе существующих. Вызов этого метода приводит к клонированию[1] каждого из элементов, содержащихся в объекте jQuery, вместе со всеми их элементами-потомками. Соответствующий пример приведен в листинге 7.2.

Листинг 7.2. Клонирование элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElems = $('div.dcell').clone();

        newElems.each(function (index, elem) {
            console.log("Новый элемент: " + elem.tagName +
                " " + elem.className);
        });

        newElems.children('img').each(function(index, elem) {
            console.log("Дочерний элемент: " + elem.tagName +
                " " + elem.src);
        });

    });
</script>
...
```

В этом сценарии выбираются и клонируются все элементы div, принадлежащие классу dcell. Для демонстрации того, что при этом клонируются также элементы-потомки, в сценарии вызывается метод children() с селектором, обеспечивающим получение клонированных элементов img. Информация об элементах div и img выводится на консоль. Результаты работы сценария выглядят следующим образом.

[1] Термин *клонирование* означает так называемое *глубокое* копирование, при котором создается не просто другая ссылка на существующий объект, а его точная копия, для хранения которой требуется тот же объем памяти, что и для оригинала. — *Примеч. ред.*

```
Новый элемент: DIV dcell
Новый элемент: DIV dcell
Новый элемент: DIV dcell
Новый элемент: DIV dcell
Новый элемент: DIV dcell
Новый элемент: DIV dcell
Дочерний элемент: IMG
http://www.jacquisflowershop.com/jquery/astor.png
Дочерний элемент: IMG
http://www.jacquisflowershop.com/jquery/daffodil.png
Дочерний элемент: IMG
http://www.jacquisflowershop.com/jquery/rose.png
Дочерний элемент: IMG
http://www.jacquisflowershop.com/jquery/peony.png
Дочерний элемент: IMG
http://www.jacquisflowershop.com/jquery/primula.png
Дочерний элемент: IMG
http://www.jacquisflowershop.com/jquery/snowdrop.png
```

Совет. Вызов метода `clone(true)` позволяет включить в процесс клонирования также обработчики событий и данные, связанные с элементами. Если аргумент опущен или имеет значение `false`, то обработчики событий и данные не копируются. О связывании данных с элементами говорится в главе 8, а о поддержке событий в jQuery — в главе 9.

Создание элементов средствами DOM API

Для создания объектов `HTMLElement` можно использовать непосредственно программный интерфейс DOM, что, собственно говоря, jQuery и делает вместо вас, когда вы привлекаете для этой цели другие методики. Я не собираюсь подробно описывать DOM API и ограничусь приведением простого примера, представленного в листинге 7.3, который даст вам возможность почувствовать, что именно представляет собой эта методика.

Листинг 7.3. Использование программного интерфейса DOM для создания элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var divElem = document.createElement("div");
        divElem.classList.add("dcell");

        var imgElem = document.createElement("img");
        imgElem.src = "lily.png";

        divElem.appendChild(imgElem);

        var newElems = $(divElem);

        newElems.each(function (index, elem) {
            console.log("Новый элемент: " + elem.tagName +
                " " + elem.className);
        });

        newElems.children('img').each(function(index, elem) {
```

```
            console.log("Дочерний элемент: " + elem.tagName +
                " " + elem.src);
        });

    });
</script>
...
```

В этом сценарии создаются и конфигурируются два объекта `HTMLElement`, представляющие HTML-элементы `div` и `img`, и элемент `img` назначается дочерним элементом элемента `div`, как это было в первом примере, приведенном в начале главы. Ничто не мешает создавать элементы таким способом, но поскольку книга посвящана jQuery, мы не будем слишком углубляться в DOM API, тем самым отклоняясь от основной темы.

Затем элемент `div` в виде объекта `HTMLElement` передается функции `$()` в качестве аргумента, что позволяет использовать в оставшейся части сценария те же самые функции, что и в предыдущих примерах. Консольный вывод будет выглядеть следующим образом.

```
Новый элемент: DIV dcell
Дочерний элемент: IMG
http://www.jacquisflowershop.com/jquery/lily.png
```

Вставка дочерних элементов и элементов-потомков

После того как вы познакомились с методами создания элементов, можно приступить к рассмотрению способов их вставки в документ. Сначала речь пойдет о методах, позволяющих вставлять одни элементы внутрь других и тем самым создавать новые дочерние элементы и другие элементы-потомки. Краткое описание этой группы методов приведено в табл. 7.2.

Таблица 7.2. Методы для вставки дочерних элементов и элементов-потомков

Метод	*Описание*

`append(HTML)`
`append(jQuery)`
`append(HTMLElement[])`
	Вставляет указанные элементы в качестве последних дочерних элементов во все выбранные элементы

`prepend(HTML)`
`preend(jQuery)`
`prepend(HTMLElement[])`
	Вставляет указанные элементы в качестве первых дочерних элементов во все выбранные элементы

`appendTo(jQuery)` `appendTo(HTMLElement[])`	Вставляет элементы, содержащиеся в объекте `jQuery`, в качестве последних дочерних элементов в элементы, заданные аргументом

`prependTo(HTML)`
`preendTo(jQuery)`
`prependTo(HTMLElement[])`
	Вставляет элементы, содержащиеся в объекте `jQuery`, в качестве первых дочерних элементов в элементы, заданные аргументом

`append(`*функция*`)` `prepend(`*функция*`)`	Добавляет результат, возвращаемый функцией, в окончание или начало содержимого каждого из элементов, содержащихся в объекте `jQuery`

Совет. Для вставки дочерних элементов можно использовать также метод `wrapInner()`, рассматриваемый далее. Этот метод помещает новый дочерний элемент между элементом и его существующими дочерними элементами. Другая возможная методика связана с использованием метода `html()`, описанного в главе 8.

Элементы, которые передаются перечисленным выше методам в виде аргументов, вставляются в качестве дочерних элементов в *каждый* выбранный элемент, содержащийся в объекте `jQuery`. В связи с этим очень важно использовать описанные в главе 6 методики, предназначенные для управления наборами выбранных элементов, таким образом, чтобы такие наборы включали лишь те элементы, с которыми вы действительно хотите работать. Пример использования метода `append()` приведен в листинге 7.4.

Листинг 7.4. Использование метода `append()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var newElems = $("<div class='dcell'></div>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />");

        newElems.css("border", "thick solid red");

        $('#row1').append(newElems);
    });
</script>
...
```

В этом сценарии метод `append()` используется двояким образом: сначала — для построения нужного набора новых элементов, а затем — для вставки этих элементов в HTML-документ. Поскольку метод `append()` — первый из рассматриваемых нами методов, предназначенных для выполнения DOM-манипуляций, я затрачу немного времени на то, чтобы продемонстрировать некоторые особенности его поведения, знание которых позволяет избежать большинства распространенных ошибок, связанных с использованием jQuery для работы с DOM. Однако сначала взгляните на результаты работы данного сценария, представленные на рис. 7.1.

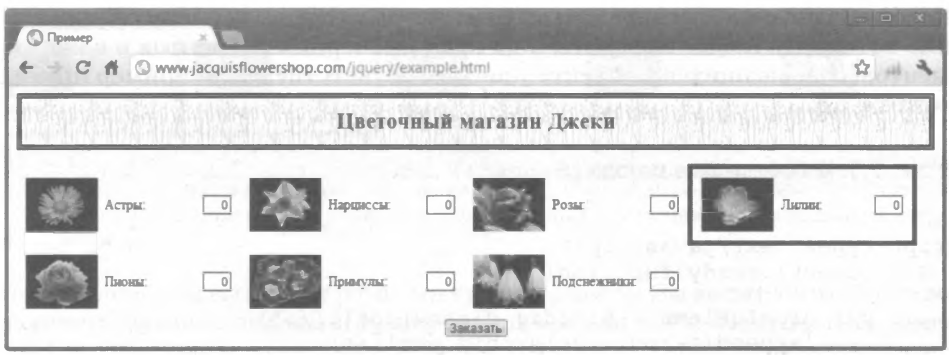


Рис. 7.1. Вставка новых элементов в документ

Первое, на что следует обратить внимание, — это способ, используемый для построения новых элементов с помощью метода `append()`.

```
var newElems = $("<div class='dcell'></div>")
    .append("<img src='lily.png'/>")
    .append("<label for='lily'>Лилии:</label>")
    .append("<input name='lily' value='0' required />");
```

Вообще говоря, можно было бы сразу создать один цельный крупный блок HTML-кода, содержащий все необходимые элементы, но мне хотелось прояснить один из ключевых аспектов методов, манипулирующих элементами DOM. Он заключается в том, что объекты `jQuery`, возвращаемые этими методами, содержат те же элементы, что и объект, вызвавший метод. Так, в данном сценарии отправной точкой служит объект `jQuery`, содержащий элемент `div`, а результатом работы каждого из вызываемых методов `append()` также является объект `jQuery()`, содержащий тот же элемент `div`, а не элемент, который добавляется. Это означает, что цепочка вызовов метода `append(0)` создает для каждого из первоначально выбранных элементов не вложенную последовательность новых элементов, а ряд новых дочерних элементов.

Следующая особенность, на которую хотелось бы обратить ваше внимание, заключается в том, что jQuery позволяет изменять вновь созданные элементы и перемещаться между ними еще до их присоединения к документу. В данном случае мы хотим выделить новые элементы рамкой, для чего используем следующий вызов:

```
newElems.css("border", "thick solid red");
```

Эта возможность чрезвычайно удобна, поскольку позволяет создавать сложные наборы элементов и в полном объеме выполнять над ними все необходимые подготовительные операции до их вставки в документ.

Наконец, вновь созданные элементы добавляются в документ с помощью следующей строки кода:

```
$('#row1').append(newElems);
```

Новые элементы добавляются в каждый из элементов выбранного набора. В данном случае выборка состоит всего лишь из одного элемента (имеющего идентификатор `row1`), добавление которого приводит к появлению на странице изображения лилии.

Вставка содержимого в начало элементов

Дополнением к методу `append()` служит метод `prepend()`, с помощью которого можно вставлять новые элементы в качестве дочерних элементов в каждый из элементов, содержащихся в объекте `jQuery`. Соответствующий пример приведен в листинге 7.5.

Листинг 7.5. Использование метода `prepend()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell'/>")
            .append("<img src='orchid.png'/>")
            .append("<label for='orchid'>Орхидеи:</label>")
            .append("<input name='orchid' value='0' required />");
```

```
        var newElems = $("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />")
            .add(orchidElems);

            newElems.css("border", "thick solid red");

            $('#row1, #row2').prepend(newElems);
        });
    </script>
...
```

В данном примере проявляется еще одна важная особенность методов jQuery, предназначенных для выполнения DOM-манипуляций: *все* элементы, передаваемые в виде аргумента любому из этих методов, добавляются в качестве дочерних элементов во *все* элементы, содержащиеся в объекте `jQuery`. В сценарии создаются два элемента `div`: один — для лилий, второй — для орхидей. Для объединения обоих наборов элементов в одном объекте jQuery используется метод `add()` (выделен в листинге).

Совет. Метод `add()` может принимать аргумент в виде строки, содержащей HTML-фрагмент. Эту возможность можно использовать в качестве альтернативного способа создания элементов с помощью объектов `jQuery`.

Далее в сценарии создается другой объект `jQuery`, содержащий элементы со значениями `id`, равными `row1` и `row1`, а вставку в документ элементов, соответствующих лилиям и орхидеям, обеспечивает вызов метода `prepend()`. Результат выполнения сценария представлен на рис. 7.2.

Рис. 7.2. Одновременное добавление нескольких новых элементов к нескольким выбранным элементам

Новые элементы выделены рамками красного цвета. Вы видите, что оба элемента, соответствующие лилиям и орхидеям, добавлены в оба элемента `row`. Вместо того чтобы использовать метод `add()`, можно воспользоваться передачей сразу нескольких элементов методам, предназначенным для изменения DOM, как показано в листинге 7.6.

Листинг 7.6. Передача нескольких элементов методу `prepend()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell'/>")
            .append("<img src='orchid.png'/>")
            .append("<label for='orchid'>Орхидеи:</label>")
            .append("<input name='orchid' value='0' required />");

        var lilyElems = $("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />");

        lilyElems.css("border", "thick solid red");

        $('#row1, #row2').prepend(lilyElems, orchidElems);
    });
</script>
...
```

Вставка одних и тех же элементов в разные места документа

Один и тот же набор новых элементов может быть вставлен в документ только один раз. После этого любая попытка использования этого набора в качестве аргумента при вызове методов, предназначенных для вставки в DOM новых узлов, приведет не к дублированию этого набора, а к его перемещению. Чтобы понять суть проблемы, обратимся к листингу 7.7.

Листинг 7.7. Двукратное добавление набора новых элементов в документ

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell'/>")
            .append("<img src='orchid.png'/>")
            .append("<label for='orchid'>Орхидеи:</label>")
            .append("<input name='orchid' value='0' required />");

        var newElems = $("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />")
            .add(orchidElems);

        newElems.css("border", "thick solid red");

        $('#row1').append(newElems);
        $('#row2').prepend(newElems);
    });
</script>
...
```

Цель данного сценария кажется очевидной: вставить набор новых элементов в окончание элемента `row1` и в начало элемента `row2`. Разумеется, на самом деле происходит нечто другое, как показано на рис. 7.3.

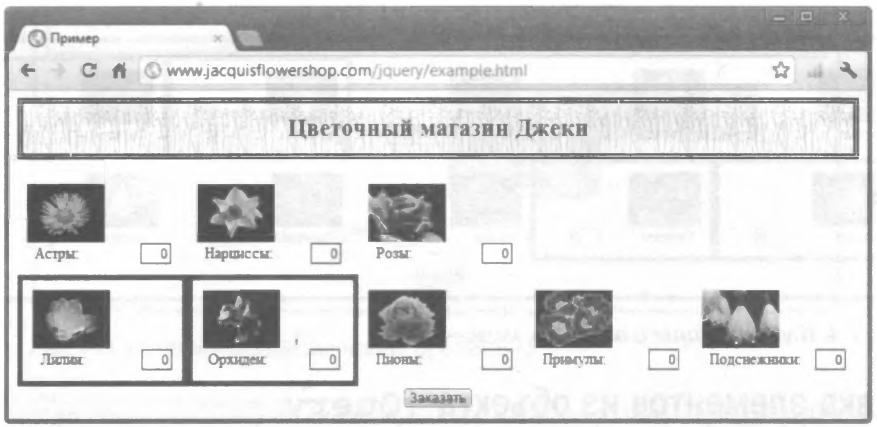


Рис. 7.3. Попытка (неудачная) дважды добавить новые элементы в документ

Действительно, элементы *вставляются* в окончание элемента `row1`, но вызов метода `prepend()` приводит не к их повторной вставке, а к перемещению. Для решения этой проблемы следует создать копии элементов, подлежащих вставке, с помощью метода `clone()`. Пересмотренный вариант сценария приведен в листинге 7.8.

Листинг 7.8. Клонирование элементов для их многократной вставки в документ

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell'/>")
            .append("<img src='orchid.png'/>")
            .append("<label for='orchid'>Орхидеи:</label>")
            .append("<input name='orchid' value='0' required />");

        var newElems = $("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />")
            .add(orchidElems);

        newElems.css("border", "thick solid red");

        $('#row1').append(newElems);
        $('#row2').prepend(newElems.clone());
    });
</script>
...
```

Теперь в результате их копирования элементы вставляются в оба места в документе (рис. 7.4).

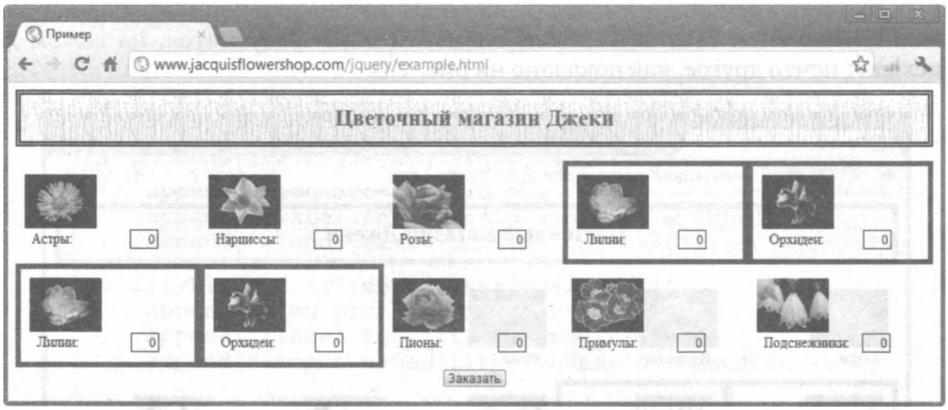


Рис. 7.4. Клонирование и вставка элементов

Вставка элементов из объекта `jQuery`

В случае использования методов `appendTo()` и `prependTo()` все меняется местами: элементы, содержащиеся в объекте `jQuery`, вставляются в качестве дочерних элементов внутрь элементов, заданных аргументом. Соответствующий пример приведен в листинге 7.9.

Листинг 7.9. Использование метода `appendTo()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElems = $("<div class='dcell'/>");

        $('img').appendTo(newElems);

        $('#row1').append(newElems);
    });
</script>
...
```

В этом сценарии мы создаем объекты `jQuery`, которые содержат новые элементы `div` и `img`, подлежащие вставке в документ. Затем элементы `img` вставляются в элемент `div` в качестве его дочерних элементов с помощью метода `appendTo()`. Конечный результат показан на рис. 7.5. Как нетрудно заметить, выполнение сценария приводит к тому, что все элементы `img` перемещаются в новый элемент `div`, который был добавлен в элемент `row1`.

Вставка элементов с использованием функции

Методы `append()` и `prepend()` могут принимать в качестве аргумента также функцию. Это обеспечивает возможность динамической вставки дочерних элементов в каждый из элементов, выбранных объектом `jQuery`, как показано в листинге 7.10.

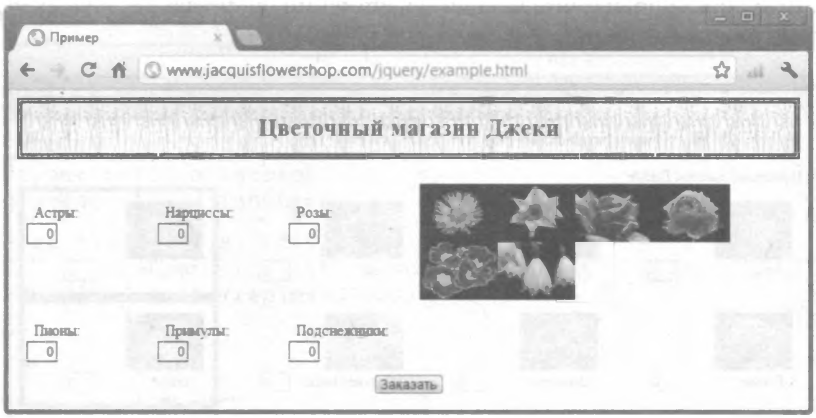


Рис. 7.5. Использование метода `appendTo()`

Листинг 7.10. Динамическое добавление дочерних элементов с помощью функции

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell'/>")
            .append("<img src='orchid.png'/>")
            .append("<label for='orchid'>Орхидеи:</label>")
            .append("<input name='orchid' value='0' required />");

        var lilyElems = $("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />");

        $(orchidElems).add(lilyElems)
            .css("border", "thick solid red");

        $('div.drow').append(function(index, html) {
            if (this.id == "row1") {
                return orchidElems;
            } else {
                return lilyElems;
            }
        });
    });
</script>
...
```

Функция вызывается однократно для каждого из элементов, содержащихся в объекте `jQuery`. Аргументами функции являются индекс элемента в выбранном наборе и строка, содержащая HTML-код обрабатываемого элемента. Переменная `this` ссылается на текущий элемент набора. Возвращаемый функцией результат вставляется в конце или в начале обрабатываемого элемента. Функция может возвращать HTML-строку, один или несколько объектов `HTMLElement` или объект `jQuery`.

В этом примере сначала создаются необходимые HTML-фрагменты для лилий и орхидей, после чего функция, передаваемая методу `append()` в качестве аргумента,

возвращает нужный фрагмент, исходя из значения свойства id. Результаты работы сценария показаны на рис. 7.6.

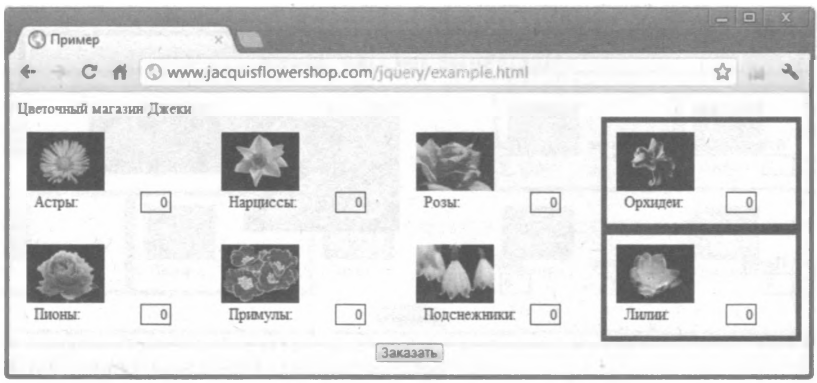


Рис. 7.6. Динамическая вставка элементов с помощью функции

Вставка родительских элементов и элементов-предков

Библиотека jQuery предоставляет набор методов, обеспечивающих вставку элементов в документ как родительских или элементов-предков по отношению к существующим элементам. Такого рода операции называются *обертыванием* (wrapping), или *внешней вставкой*, (поскольку добавляемый элемент окружает собой уже существующие элементы). Краткое описание этой группы методов приведено в табл. 7.3.

Таблица 7.3. Методы для внешней вставки родительских элементов и элементов-предков

Метод	Описание

`wrap(HTML)`
`wrap(jQuery)`
`wrap(HTMLElement[])`
	Обертывает указанные элементы вокруг каждого из элементов, содержащихся в объекте `jQuery`

`wrapAll(HTML)`
`wrapAll(jQuery)`
`wrapAll(HTMLElement[])`
	Обертывает указанные элементы вокруг набора элементов, содержащихся в объекте `jQuery` (рассматриваемых как единая группа)

`wrapInner(HTML)`
`wrapInner(jQuery)`
`wrapInner(HTMLElement[])`
	Обертывает указанные элементы вокруг содержимого каждого из элементов, содержащихся в объекте `jQuery`

`wrap(`*функция*`)` `wrapInner(`*функция*`)`	Динамически обертывает элементы с использованием функции

При обертывании элементов эти методы могут принимать HTML-фрагмент в качестве аргумента, но всегда следует проверять, чтобы этот фрагмент содержал только один внутренний элемент. В противном случае jQuery не сможет определить, что именно необходимо сделать. Отсюда следует, что каждый элемент в аргументе метода может иметь не более одного родительского и не более одного дочернего элемента. Пример использования метода `wrap()` приведен в листинге 7.11.

Совет. Перечисленные методы дополняет метод `unwrap()`, о котором говорится ниже.

Листинг 7.11. Использование метода `wrap()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>")
            .css("border", "thick solid red");
        $('div.drow').wrap(newElem);

    });
</script>
...
```

В этом сценарии мы создаем новый элемент `div` и используем метод `css()` для установки значения CSS-свойства `border`. Затем элемент `div` добавляется в качестве родительского элемента по отношению ко всем элементам `label` в документе. Результаты выполнения сценария представлены на рис. 7.7.

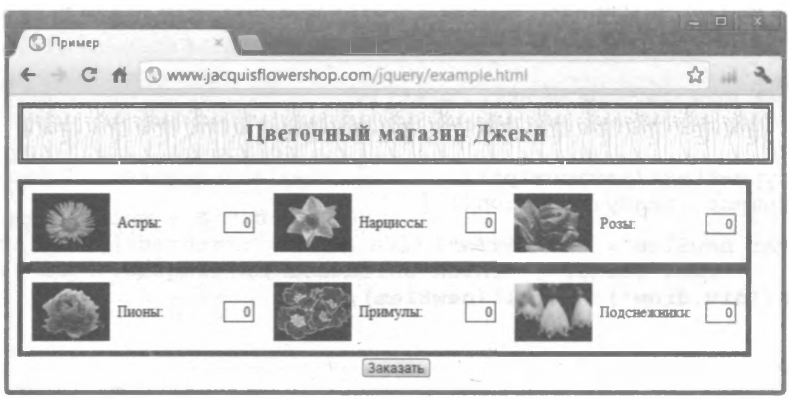


Рис. 7.7. Использование метода `wrap()` для добавления родительских элементов

Элементы, которые вы передаете в виде аргумента методу `wrap()`, вставляются между каждым из элементов, содержащихся в объекте `jQuery`, и его текущим родительским элементом. Например, следующий HTML-фрагмент:

```
...
<div class="dtable">
    <div id="row1" class="drow">
        ...
    </div>
    <div id="row2" class="drow">
        ...
    </div>
</div>
...
```

будет преобразован в такой фрагмент:

```
...
<div class="dtable">
    <div style="...стилевые свойства...">
        <div id="row1" class="drow">
            ...
        </div>
    </div>
    <div style="...стилевые свойства...">
        <div id="row2" class="drow">
        ...
        </div>
    </div>
</div>
...
```

Обертывание набора элементов

Когда используется метод `wrap()`, новые элементы клонируются, и каждый из элементов, содержащихся в объекте `jQuery`, получает собственный новый родительский элемент. Чтобы вставить в документ элемент, который станет родительским сразу для нескольких элементов, следует использовать метод `wrapAll()`, как показано в листинге 7.12.

Листинг 7.12. Использование метода `wrapAll()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>")
            .css("border", "thick solid red");
        $('div.drow').wrapAll(newElem);

    });
</script>
...
```

Этот сценарий отличается от предыдущего лишь тем, что в нем используется метод `wrapAll()`. Страница в окне браузера показана на рис. 7.8.

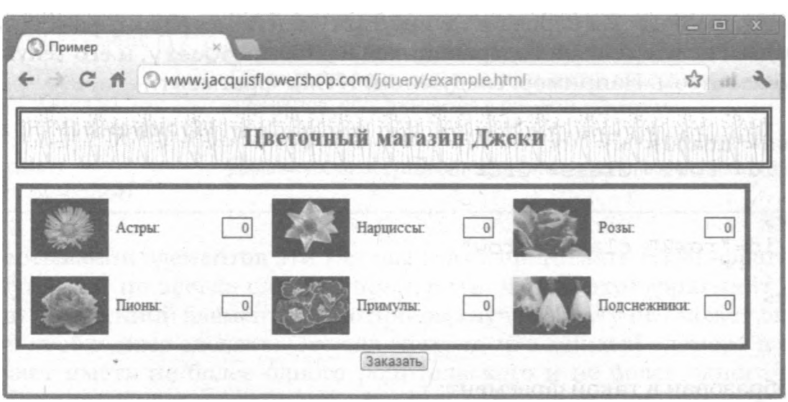


Рис. 7.8. Использование метода `wrapAll()`

Новый элемент добавляется как общий родительский элемент для всего набора выбранных элементов, в результате чего HTML-разметка преобразуется следующим образом.

```
<div class="dtable">
    <div style="...стилевые свойства...">
        <div id="row1" class="drow">
            ...
        </div>
        <div id="row2" class="drow">
        ...
        </div>
    </div>
</div>
...
```

Будьте внимательны, используя этот метод. Если выбранные элементы не имеют общих родителей, то новый элемент будет добавлен как родительский для первого из них. После этого jQuery переместит все остальные элементы так, чтобы они стали сестринскими по отношению к первому элементу. Пример использования метода `wrapAll()` приведен в листинге 7.13.

Листинг 7.13. Применение метода `wrapAll()` к элементам, не имеющим общего родителя

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>")
            .css("border", "thick solid red");
        $('img').wrapAll(newElem);

    });
</script>
...
```

В этом сценарии выбираются элементы `img`, и эти элементы не имеют общих родителей. Вид страницы в окне браузера представлен на рис. 7.9.

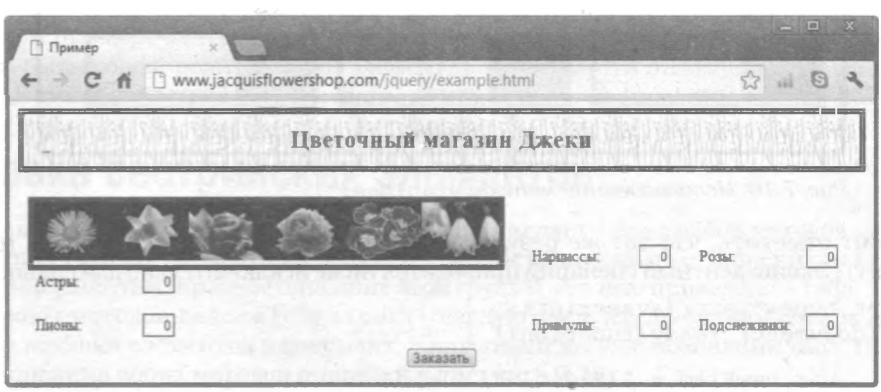


Рис. 7.9. Использование метода `wrapAll()` в случае элементов, не имеющих общего родителя

Новый элемент `div` был добавлен в документ как родительский по отношению к элементу `img`, соответствующему изображению астры, тогда как все остальные элементы изображений были перемещены в структуре DOM в позиции сестринских по отношению к нему элементов.

Обертывание содержимого элементов

Метод `wrapInner()` позволяет окружать элементами содержимое других элементов (а не сами элементы). Соответствующий пример приведен в листинге 7.14.

Листинг 7.14. Использование метода `wrapInner()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>")
            .css("border", "thick solid red");
        $('.dcell').wrapInner(newElem);

    });
</script>
...
```

Метод `wrapInner()` вставляет новые элементы между каждым из содержащихся в объекте `jQuery` элементом и его дочерним элементом. В данном сценарии выбираются элементы, принадлежащие классу `dcell`, и содержимое каждого из них заключается в новый элемент `div`. Страница в окне браузера показана на рис. 7.10.

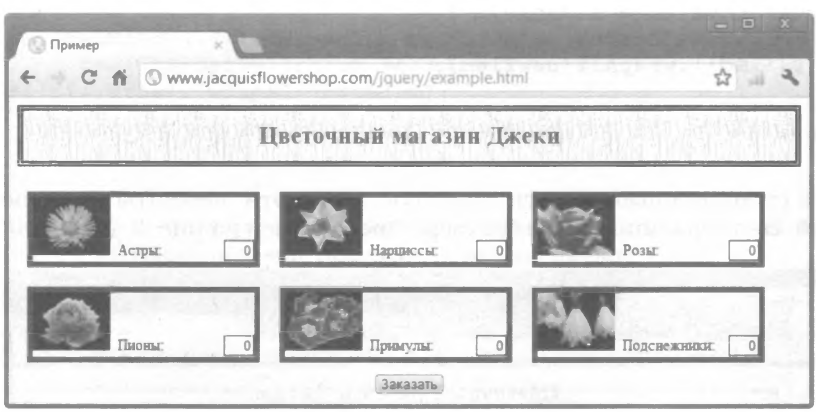


Рис. 7.10. Использование метода `wrapInner()`

Стоит отметить, что тот же результат может быть получен с помощью метода `append()`. Эквивалентный сценарий приводится ниже исключительно для сравнения.

```
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>")
            .css("border", "thick solid red");
        $('.dcell').each(function(index. elem) {
        $(elem).append(newElem.clone()
```

```
            .append($(elem).children()));
        });

    });
</script>
...
```

Я не рекомендую применять такой подход (использование метода `wrapInner()` упрощает работу и облегчает чтение кода), но считаю, что это хороший пример того, как jQuery позволяет решать одну и ту же задачу самыми разными способами.

Обертывание элементов с использованием функции

Методам `wrap()` и `wrapAll()` в качестве аргумента может быть передана функция, что обеспечивает возможность динамической генерации элементов. Единственным аргументом этой функции является индекс элемента в выбранном наборе. Внутри этой функции специальная переменная `this` ссылается на обрабатываемый элемент. Пример использования метда `wrap()` в режиме динамической генерации элементов приведен в листинге 7.15.

Листинг 7.15. Динамическое обертывание элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('.drow').wrap(function(index) {
            if ($(this).has('img[src*=rose]').length > 0) {
                return $("<div/>")
                .css("border", "thick solid blue");;
            } else {
                return $("<div/>")
                .css("border", "thick solid red");;
            }
        });

    });
</script>
...
```

В этом примере методу `wrap()` передается функция, которая определяет, каким должен быть новый родительский элемент, в зависимости от выбранного элемента, который она обрабатывает. Вид страницы в окне браузера представлен на рис. 7.11.

Вставка сестринских элементов

Как вы уже могли догадаться, jQuery предоставляет также набор методов, обеспечивающих вставку элементов в документ как сестринских по отношению к существующим элементам. Краткое описание этой группы методов приведено в табл. 7.4.

Вызовы методов `before()` и `after()` следуют тем же шаблонам, что и другие методы вставки элементов в документ, с которыми вы уже познакомились. Пример использования обоих методов приведен в листинге 7.16.

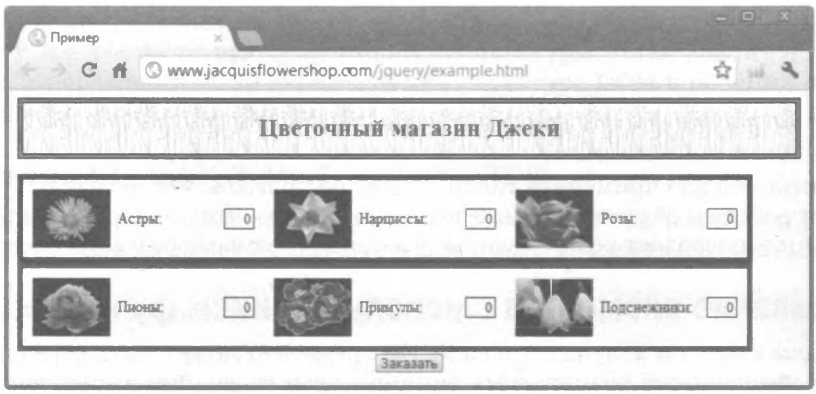


Рис. 7.11. Использование метода `wrap()` с функцией для динамической генерации родительских элементов

Таблица 7.4. Методы для вставки сестринских элементов

Метод	*Описание*

`after(HTML)`
`after(jQuery)`
`after(HTMLElement[])`
	Вставляет указанные элементы в качестве последних дочерних элементов во все выбранные элементы

`before(HTML)`
`before(jQuery)`
`before(HTMLElement[])`
	Вставляет указанные элементы в качестве первых дочерних элементов во все выбранные элементы

`insertAfter(HTML)`
`insertAfter(jQuery)`
`insertAfter(HTMLElement[])`
	Вставляет элементы, содержащиеся в объекте `jQuery`, в качестве последних дочерних элементов в элементы, заданные аргументом

`insertBefore(HTML)`
`insertBefore(jQuery)`
`insertBefore(HTMLElement[])`
	Вставляет элементы, содержащиеся в объекте `jQuery`, в качестве первых дочерних элементов в элементы, заданные аргументом

`after(`*функция*`)` `before(`*функция*`)`	Добавляет результат, возвращаемый функцией, в окончание или начало содержимого каждого из элементов, содержащихся в объекте `jQuery`

Листинг 7.16. Использование методов `before()` и `after()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell'/>")
            .append("<img src='orchid.png'/>")
            .append("<label for='orchid'>Орхидеи:</label>")
            .append("<input name='orchid' value='0' required />");

        var lilyElems = $("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />");
```

```
        $(orchidElems).add(lilyElems).css("border", "thick solid red");

        $('#row1 div.dcell').after(orchidElems);
        $('#row2 div.dcell').before(lilyElems);

    });
</script>
...
```

В этом сценарии мы создаем наборы элементов для орхидей и лилий, а затем вставляем их в документ с помощью методов `before()` и `after()` как сестринские элементы по отношению к каждому из элементов, принадлежащих классу `dcell`. Элементы, соответствующие орхидеям, вставляются как следующие сестринские элементы по отношению к каждому элементу в контейнере со значением `id`, равным `row1`, тогда как элементы, соответствующие лилиям, вставляются как предшествующие сестринские элементы по отношению к каждому элементу в контейнере со значением `id`, равным `row2`. Вид страницы в окне браузера представлен на рис. 7.12.

Рис. 7.12. Использование методов `before()` и `after()` для создания сестринских элементов

Вставка сестринских элементов из объекта `jQuery`

Методы `insertAfter()` и `insertBefore()` вставляют элементы, содержащиеся в объекте `jQuery`, в качестве следующих или предшествующих сестринских элементов по отношению к элементам, переданным методу в качестве аргумента. Это та же функциональность, которую обеспечивают методы `after()` и `before()`, но отличающаяся тем, что объект `jQuery` и аргумент метода меняются ролями. Пример использования обоих методов приведен в листинге 7.17.

Листинг 7.17. Использование методов `insertAfter()` и `insertBefore()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell'/>")
            .append("<img src='orchid.png'/>")
            .append("<label for='orchid'>Орхидеи:</label>")
            .append("<input name='orchid' value='0' required />");
```

```
        var lilyElems = $("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />");

        $(orchidElems).add(lilyElems).css("border", "thick solid
red");

        orchidElems.insertAfter('#row1 div.dcell');
        lilyElems.insertBefore('#row2 div.dcell');
    });
</script>
...
```

Вставка сестринских элементов с использованием функции

Сестринские элементы можно вставлять динамически, передавая методам `after()` и `before()` функцию в качестве аргумента, как мы это уже делали при вставке родительских и дочерних элементов. Пример динамической генерации сестринских элементов приведен в листинге 7.18.

Листинг 7.18. Динамическая генерация сестринских элементов с использованием функции

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#row1 div.dcell').after(function(index, html) {
            if (index == 0) {
                return $("<div class='dcell'/>")
                    .append("<img src='orchid.png'/>")
                    .append("<label for='orchid'>Орхидеи:
                        </label>")
                    .append("<input name='orchid'
                        value='0' ired />")
                    .css("border", "thick solid red");
            } else if (index == 1) {
                return $("<div class='dcell'/>")
                    .append("<img src='lily.png'/>")
                    .append("<label for='lily'>Лилии:
                        </label>")
                    .append("<input name='lily'
                        value='0' required />")
                    .css("border", "thick solid red");
            }
        });

    });
</script>
...
```

В этом сценарии сестринские элементы генерируются для тех элементов, для которых передаваемый функции аргумент `index` принимает значение `0` или `1`. Страница в окне браузера показана на рис. 7.13.

Рис. 7.13. Добавление сестринских элементов с использованием функции

Замена элементов

С помощью методов, описанных в табл. 7.5, можно заменить один набор элементов другим.

Таблица 7.5. Методы замены элементов

Метод	Описание

`replaceWith(HTML)`
`replaceWith(jQuery)`
`replaceWith(HTMLElement[])`
	Заменяет элементы, содержащиеся в объекте `jQuery`, указанным содержимым

`replaceAll(jQuery)` `replaceAll(HTMLElement[])`	Заменяет элементы, заданные аргументом, элементами, содержащимися в объекте `jQuery`
`replaceWith(`*функция*`)`	Выполняет динамическую замену элементов, содержащихся в объекте `jQuery`, с использованием функции

Методы `replaceWith()` и `replaceAll()` работают одинаковым образом, за исключением того, что объект `jQuery` и аргумент играют в них противоположные роли. Пример использования обоих методов приведен в листинге 7.19.

Листинг 7.19. Использование методов `replaceWith()` и `replaceAll`

```
<script type="text/javascript">
    $(document).ready(function() {

        var newElems = $("<div class='dcell'/>")
                    .append("<img src='orchid.png'/>")
                    .append("<label for='orchid'>Орхидеи:
                        </label>")
                    .append("<input name='orchid' value='0'
                        required />")
                    .css("border", "thick solid red");

        $('#row1').children().first().replaceWith(newElems);

        $("<img src='carnation.png'/>").replaceAll('#row2 img')
            .css("border", "thick solid red");
```

```
    });
</script>
...
```

В этом сценарии сначала создается набор элементов, а затем в документе выполняется поиск элемента `div`, атрибут `id` которого равен `row1`, и его первый дочерний элемент заменяется новым содержимым с помощью метода `replaceWith()` (что в конечном счете приводит к замене элементов, соответствующих астрам, элементами, соответствующими орхидеям). Наконец, с помощью метода `replaxeAll()` все элементы `img`, являющиеся потомками элемента, атрибут `id` которого равен `row2`, заменяются изображениями гвоздики. Вид страницы в окне браузера представлен на рис. 7.14.

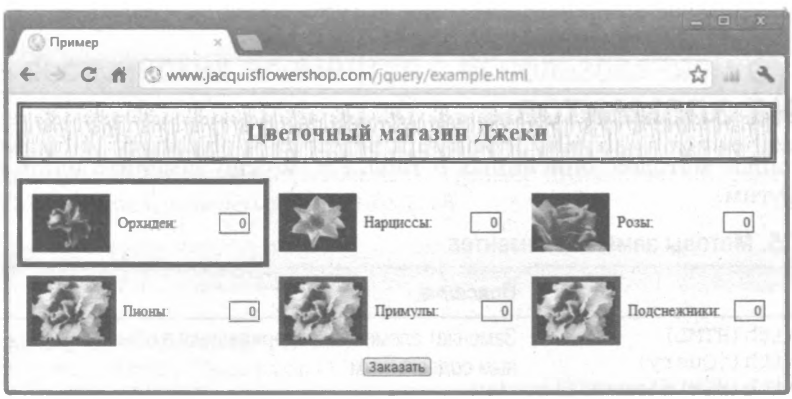


Рис. 7.14. Замена содержимого с помощью методов `replaceWith()` и `replaceAll()`

Замена элементов с использованием функции

Передавая функцию методу `replaceWith()`, можно выполнять динамическую замену элементов. Аргументы этой функции не передаются, однако переменная `this` внутри нее ссылается на обрабатываемый элемент. Соответствующий пример приведен в листинге 7.20.

Листинг 7.20. Замена элементов с использованием функции

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('div.drow img').replaceWith(function() {
            if (this.src.indexOf("rose") > -1) {
                return $("<img src='carnation.png'/>")
                    .css("border", "thick solid red");
            } else if (this.src.indexOf("peony") > -1) {
                return $("<img src='lily.png'/>")
                .css("border", "thick solid red");
            } else {
                return $(this).clone();
            }
        });
```

```
    });
</script>
...
```

В этом сценарии мы выполняем замену элементов `img` на основании значений их атрибутов `src`. Если этот атрибут элемента `img` содержит `rose`, то данный элемент заменяется другим, которому соответствует изображение `carnation.png`. Если же атрибут `src` элемента содержит `peony`, то данный элемент заменяется другим, которому соответствует изображение `lily.png`. Оба замененных элемента выделяются рамкой красного цвета, чтобы сделать эффект более заметным. Страница в окне браузера показана на рис. 7.15.

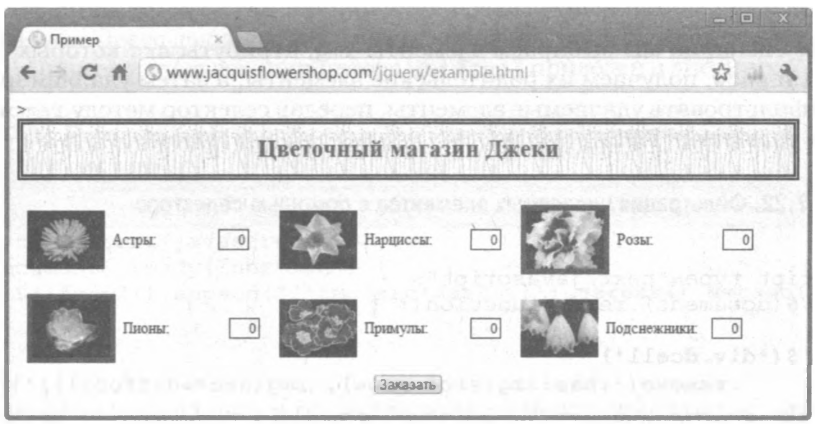


Рис. 7.15. Замена элементов с использованием функции

Совет. Если замена элемента нежелательна, можете просто вернуть его клон. Если не клонировать элемент, то jQuery полностью удалит его из документа. Конечно, этой проблемы можно избежать путем усечения выбранного набора, но такая возможность не всегда имеется.

Удаление элементов

В дополнение к возможности вставки и замены элементов библиотека jQuery предлагает ряд методов для удаления элементов из DOM. Краткое описание этой группы методов приведено в табл. 7.6.

Таблица 7.6. Методы для удаления элементов

Метод	Описание
`detach()` `detach(`*селектор*`)`	Удаляет элементы из DOM. Данные, связанные с элементами, сохраняются
`empty()`	Удаляет все дочерние узлы каждого из элементов, содержащихся в объекте `jQuery`
`remove()` `remove(`*селектор*`)`	Удаляет элементы из DOM. По мере удаления элементов связанные с ними данные уничтожаются
`unwrap()`	Удаляет родительские элементы каждого из элементов, содержащихся в объекте `jQuery`

Пример использования метода `remove()` для удаления элементов приведен в листинге 7.21.

Листинг 7.21. Удаление элементов из DOM с помощью метода `remove()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img[src*=daffodil], img[src*=snow]').parent().remove();
    });
</script>
...
```

В этом сценарии мы выбираем элементы `img`, атрибуты `src` которых содержат `daffodil` и `snow`, получаем их родительские элементы, а затем удаляем их. Можно также отфильтровать удаляемые элементы, передав селектор методу `remove()`, как показано в листинге 7.22.

Листинг 7.22. Фильтрация удаляемых элементов с помощью селектора

```
    <script type="text/javascript">
        $(document).ready(function() {

        $('div.dcell')
            .remove(':has(img[src*=snow], img[src*=daffodil])');

    });
</script>
...
```

Оба сценария приводят к одному и тому же результату, показанному на рис. 7.16.

Совет. Как показывает мой опыт работы с методом `remove()`, не все селекторы способны выполнять функцию фильтра. Я рекомендую тщательно тестировать каждый шаг и отдавать предпочтение исходному набору выбранных элементов, если только это возможно.

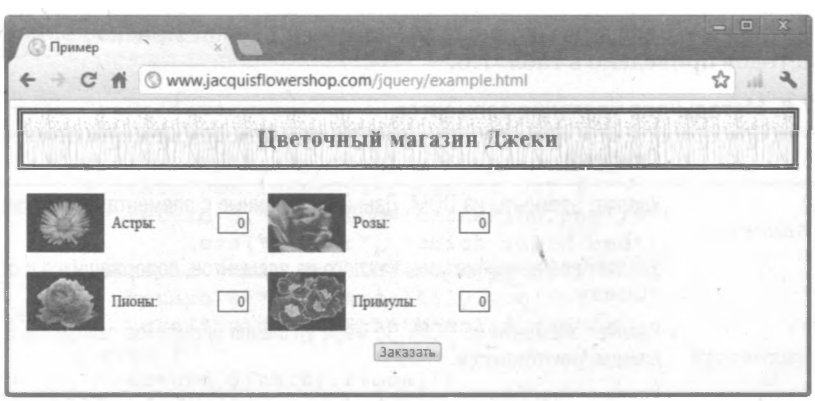


Рис. 7.16. Удаление элементов из DOM

Совет. Возвращаемый методом `remove()` объект `jQuery` содержит исходный набор выбранных элементов. Иными словами, удаление элементов не отражается на результате, который возвращает метод.

Удаление элементов с сохранением данных

Метод `detach()` работает аналогично методу `remove()` с тем лишь отличием, что связанные с удаляемыми элементами данные сохраняются. О связывании данных с элементами подробно говорится в главе 8, а на данном этапе вам достаточно знать лишь то, что если планируется последующая вставка удаленных элементов в другое место документа, то обычно предпочтение следует отдавать методу `detach()`. Пример использования метода `detach()` приведен в листинге 7.23.

Листинг 7.23. Использование метода `detach()` для удаления элементов с сохранением связанных с ними данных

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#row2').append($('img[src*=astor]').parent().detach());
    });
</script>
...
```

В этом сценарии удаляется родительский элемент элемента `img`, атрибут `src` которого содержит `astor`. Затем элементы вновь вставляются в документ с помощью рассмотренного нами ранее метода `append()`. Я стараюсь избегать такого подхода, поскольку использование метода `append()` без вызова метода `detach()` дает тот же эффект. Инструкцию, являющуюся ключевой в листинге, можно переписать следующим образом:

```
$('#row2').append($('img[src*=astor]').parent());
```

Страница в окне браузера показана на рис. 7.17.

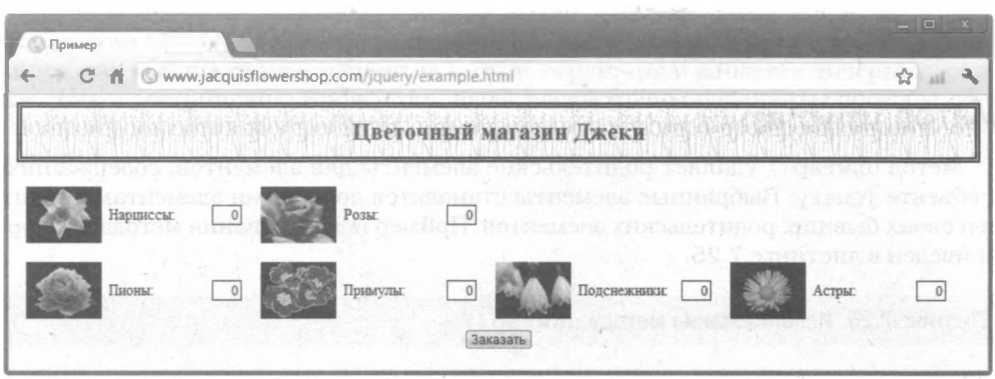


Рис. 7.17. Использование метода `detach()`

Очистка элементов

Метод `empty()` удаляет все элементы-потомки и текст из элементов, содержащихся в объекте `jQuery`. Сами элементы остаются в документе. Соответствующий пример приведен в листинге 7.24.

Листинг 7.24. Использование метода `empty()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#row1').children().eq(1).empty()
            .css("border", "thick solid red");
    });
</script>
...
```

В этом сценарии из дочерних элементов элемента с атрибутом `id`, равным `row1`, выбирается элемент с индексом, равным `1`, и вызывается метод `empty()`. Чтобы сделать изменение более заметным, соответствующая позиция в документе заключена в красную рамку. Страница в окне браузера показана на рис. 7.18.

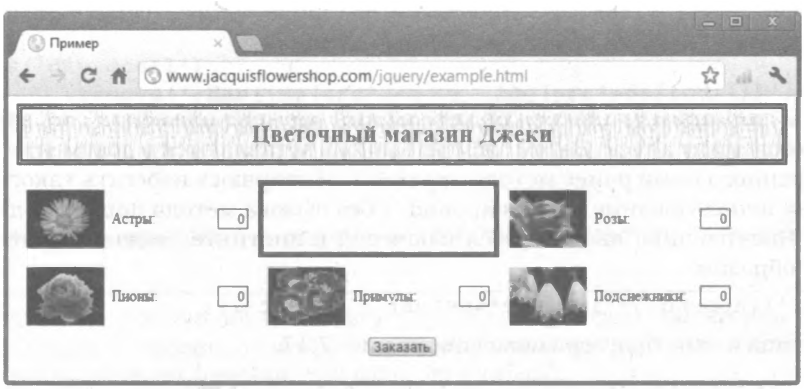


Рис. 7.18. Использование метода `empty()`

Метод `unwrap()`

Метод `unwrap()` удаляет родительские элементы для элементов, содержащихся в объекте `jQuery`. Выбранные элементы становятся дочерними элементами родителей своих бывших родительских элементов. Пример использования метода `unwrap()` приведен в листинге 7.25.

Листинг 7.25. Использование метода `unwrap()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#row1 div').unwrap().css({'display':'block'});
    });
```

```
</script>
...
```

В этом сценарии выбираются элементы `div`, являющиеся потомками элемента с атрибутом `id`, равным `row1`, и вызывается метод `unwrap()`, что приводит к удалению элемента `row1`, как показано на рис. 7.19. Изменение расположения на странице элементов, лишенных своих прежних родительских элементов, обусловлено тем, что в определение используемых стилей CSS, обеспечивающих расположение элементов в виде таблицы, входит идентификатор `row1`.

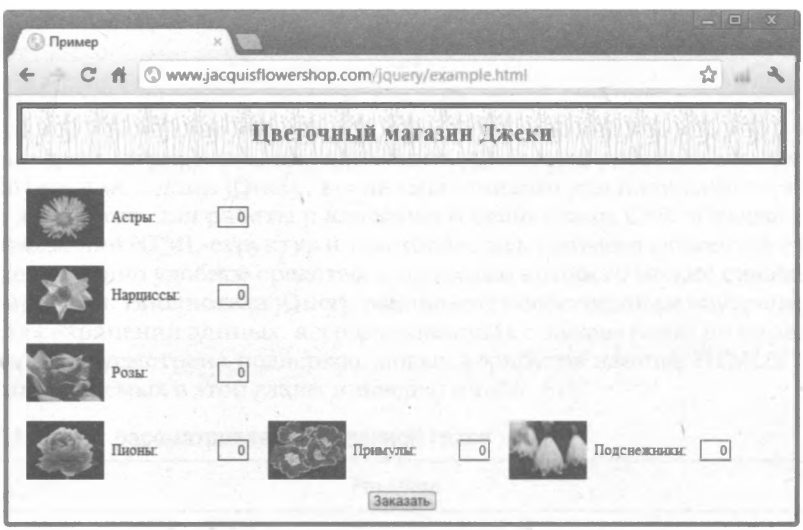


Рис. 7.19. Использование метода `unwrap()`

Резюме

Из этой главы вы узнали, как использовать средства jQuery для манипуляций элементами DOM. Мы рассмотрели методы jQuery, предназначенные как для создания, так и для вставки элементов в DOM-структуру в качестве дочерних, родительских и сестринских. Кроме того, были рассмотрены различные способы перемещения элементов внутри DOM или их удаления, если в этом возникает необходимость.

ГЛАВА 8

Манипуляции элементами

В этой главе демонстрируется, как применять jQuery для работы с элементами. Вы познакомитесь с методами jQuery, предназначенными для получения и изменения значений атрибутов, для работы с классами и свойствами CSS, а также для получения и изменения HTML-структур и текстового содержимого элементов страницы. Также будет описано удобное средство, с помощью которого можно связывать данные с элементами. Библиотека jQuery располагает собственным внутренним механизмом для сохранения данных, ассоциированных с элементами, но наряду с этим в ней также предусмотрена поддержка новых атрибутов данных HTML5. Перечень тем, рассматриваемых в этой главе, приведен в табл. 8.1.

Таблица 8.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Получение значения атрибута первого из элементов, содержащихся в объекте `jQuery`	Используйте метод `attr()`	1
Получение значения атрибута каждого из элементов, содержащихся в объекте `jQuery`	Используйте совместно методы `each()` и `attr()`	2
Изменение значения атрибута для всех элементов, содержащихся в объекте `jQuery`	Используйте метод `attr()`, возможно — с функцией	3
Изменение значения нескольких атрибутов в рамках одной операции	Используйте метод `attr()` с объектом отображения	4, 5
Сброс атрибута	Используйте метод `removeAttr()`	6
Получение или изменение значения свойства, определенного объектом `HTMLElement`	Используйте соответствующие варианты метода `attr()`	7
Управление классами, которым принадлежат элементы	Используйте методы `addClass()`, `hasClass()` и `removeClass()`, возможно — с функцией	8–10
Переключение классов, которым принадлежат элементы	Используйте метод `toggleClass()`	11–16
Изменение содержимого атрибута `style`	Используйте метод `css()`	17–21
Получение подробной информации о позициях элементов	Используйте методы, специфические для свойств CSS	22, 23
Получение или изменение текста или HTML-содержимого элементов	Используйте метод `text()` или `html()`	24–26

Окончание табл. 8.1

Задача	Решение	Листинг
Получение или изменение значений элементов формы	Используйте метод `val()`	27–29
Связывание данных с элементами	Используйте метод `data()`	30, 31

Работа с атрибутами и свойствами

Библиотека jQuery позволяет получать и устанавливать значения атрибутов элементов, содержащихся в объекте `jQuery`. Методы jQuery, предназначенные для работы с атрибутами, описаны в табл. 8.2.

Таблица 8.2. Методы для работы с атрибутами

Метод	Описание
`attr(`*имя*`)`	Возвращает значение атрибута с указанным именем для первого из элементов, содержащихся в объекте `jQuery`
`attr(`*имя, значение*`)`	Устанавливает значение атрибута с указанным именем для всех элементов, содержащихся в объекте `jQuery`
`attr(`*отображение*`)`	Устанавливает атрибуты, указанные в объекте отображения, для всех элементов, содержащихся в объекте `jQuery`
`attr(`*имя, функция*`)`	Устанавливает указанный атрибут для всех элементов, содержащихся в объекте `jQuery`, с помощью функции
`removeAttr(`*имя*`)` `removeAttr(`*имя[]*`)`	Удаляет атрибут (атрибуты) из всех элементов, содержащихся в объекте `jQuery`
`prop(`*имя*`)`	Возвращает значение указанного свойства для первого из элементов, содержащихся в объекте `jQuery`
`prop(`*имя, значение*`)` `prop(`*отображение*`)`	Устанавливает значение одного или нескольких свойств для всех элементов, содержащихся в объекте `jQuery`
`prop(`*имя, функция*`)`	Устанавливает значение указанного свойства для всех элементов, содержащихся в объекте `jQuery`, с использованием функции
`removeProp(`*имя*`)`	Удаляет указанное свойство из всех элементов, содержащихся в объекте `jQuery`

Если метод `attr()` вызывается с одним аргументом, jQuery возвращает значение указанного атрибута для первого из элементов выбранного набора. Соответствующий пример приведен в листинге 8.1.

Листинг 8.1. Чтение значения атрибута

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
```

```
    <script type="text/javascript">
        $(document).ready(function() {
            var srcValue = $('img').attr('src');
            console.log("Значение атрибута: " + srcValue);
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
           <script type="text/javascript">
        $(document).ready(function() {
            var srcValue = $('img').attr('src');
            console.log("Значение атрибута: " + srcValue);
        });
    </script> </div>
```

```
    </form>
</body>
</html>
```

В этом сценарии мы выбираем все элементы `img` в документе, а затем используем метод `attr()` для получения значения атрибута `src`, при считывании которого получаем строку. На консоль выводится следующий результат.

```
Значение атрибута: astor.png
```

Получение значения атрибута для всех элементов, содержащихся в объекте `jQuery`, обеспечивается совместным применением методов `each()` и `attr()`. Метод `each()` описан в главе 5, а пример его использования в данной ситуации приведен в листинге 8.2.

Листинг 8.2. Использование методов `each()` и `attr()` для получения значений атрибутов группы объектов

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img').each(function(index, elem) {
            var srcValue = $(elem).attr('src');
            console.log("Значение атрибута: " + srcValue);
        });
    });
</script>
...
```

В этом сценарии из объекта `HTMLElement`, передаваемого функции в качестве аргумента, с помощью функции `$()` создается объект `jQuery`. Этот объект содержит единственный элемент, который идеально подходит для метода `attr()`. Результат, выводимый на консоль, будет иметь следующий вид.

```
Значение атрибута: astor.png
Значение атрибута: daffodil.png
Значение атрибута: rose.png
Значение атрибута: peony.png
Значение атрибута: primula.png
Значение атрибута: snowdrop.png
```

Установка значений атрибутов

Если метод `attr()` используется для установки значения атрибута, то изменение применяется ко всем элементам, содержащимся в объекте `jQuery`. Таким образом, в данном случае метод ведет себя иначе, чем при считывании атрибутов, когда возвращается лишь значение атрибута одного элемента. При установке атрибута метод `attr()` возвращает объект `jQuery`, что означает возможность использования цепочки вызовов. Пример сценария, выполняющего установку значения атрибута, приведен в листинге 8.3.

Листинг 8.3. Установка атрибута

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img').attr("src", "lily.png");
    });
</script>
...
```

В этом сценарии выбираются все элементы `img` в документе, и для атрибута `src` устанавливается значение `lily.png`. Установленное значение применяется ко всем выбранным элементам, как показано на рис. 8.1.

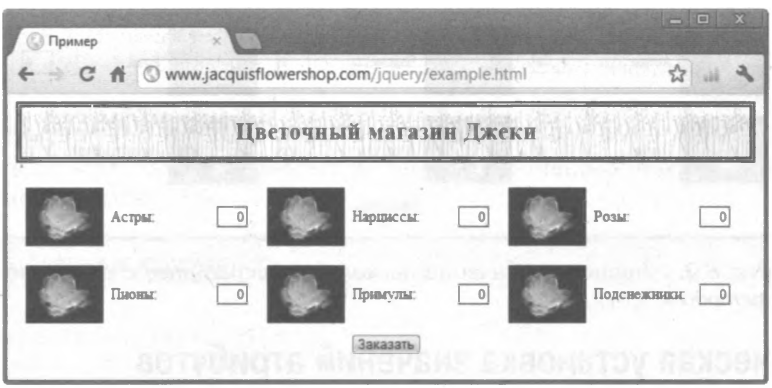


Рис. 8.1. Установка одного и того же значения атрибута для нескольких элементов

Установка нескольких атрибутов

Можно установить значения нескольких атрибутов одним вызовом метода `attr()`, передав ему в качестве аргумента объект. Свойства такого объекта интерпретируются как имена атрибутов, а значения свойств — как значения атрибута. Этот объект принято называть *объектом отображения* (map object). Соответствующий пример приведен в листинге 8.4.

Листинг 8.4. Установка нескольких атрибутов с использованием объекта отображения

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var attrValues = {
            src: 'lily.png',
            style: 'border: thick solid red'
        };

        $('img').attr(attrValues);
    });
</script>
...
```

В этом сценарии создается объект, определяющий свойства с именами `src` и `style`. Далее в документе выбираются элементы `img`, и методу `attr()` передается объект отображения. Вид страницы в окне браузера приведен на рис. 8.2.

Совет. В данном примере свойство `style` устанавливается явным образом, однако в jQuery имеется ряд методов, упрощающих работу с CSS. Далее этот вопрос будет обсуждаться более подробно.

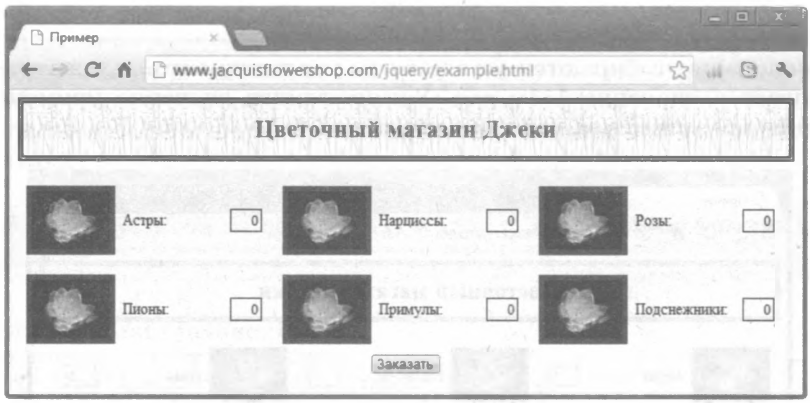


Рис. 8.2. Установка значений нескольких атрибутов с помощью метода `attr()`

Динамическая установка значений атрибутов

Назначаемые атрибутам значения можно определять во время выполнения сценария, передавая методу `attr()` функцию в качестве аргумента. Соответствующий пример приведен в листинге 8.5.

Листинг 8.5. Установка значений атрибутов с помощью функции

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img').attr("src", function(index, oldVal) {
            if (oldVal.indexOf("rose") > -1) {
                return "lily.png";
            } else if ($(this).closest('#row2').length > 0) {
                return "carnation.png";
            }
        });
    });
</script>
...
```

Аргументы, передаваемые указанной функции, — это индекс обрабатываемого элемента в наборе и прежнее значение атрибута. Переменная `this` ссылается на текущий обрабатываемый объект `HTMLElement`. Если вы хотите изменить атрибут, то функция должна вернуть строку, содержащую новое значение. Если же результат не возвращается, то используется прежнее значение атрибута. В данном примере функция используется для избирательного изменения изображений, отображаемых элементами `img`. Вид страницы в окне браузера приведен на рис. 8.3.

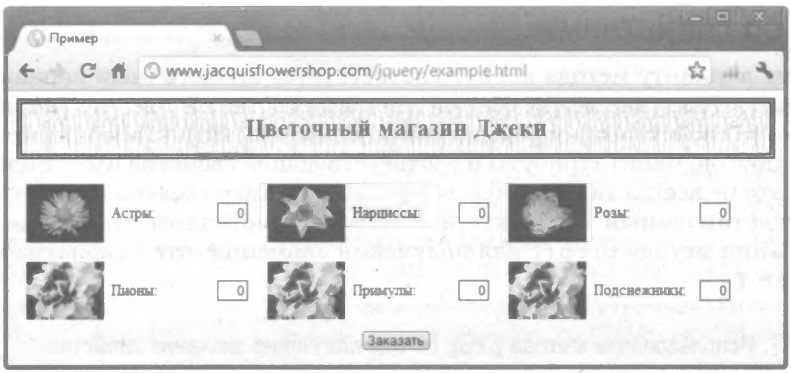


Рис. 8.3. Изменение значений атрибутов с помощью функции

Удаление атрибутов

Атрибуты можно удалять (отменять установку) с помощью метода `removeAttr()`, как показано в листинге 8.6.

Листинг 8.6. Удаление атрибутов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').attr("style", "border: thick solid red");
        $('img:odd').removeAttr("style");

    });
</script>
...
```

В этом примере атрибут `style` сначала устанавливается с помощью метода `attr()`, а затем удаляется из нечетных элементов `img` с помощью метода `removeAttr()`. Вид страницы в окне браузера приведен на рис. 8.4.

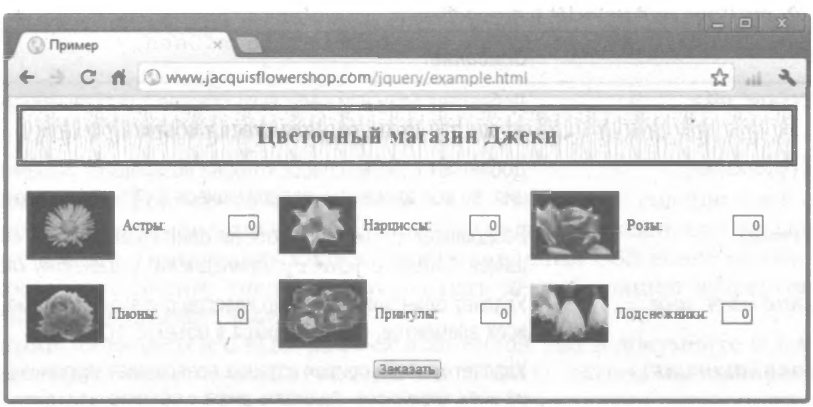


Рис. 8.4. Удаление атрибутов из элементов

Работа со свойствами

Каждому варианту метода `attr()` соответствует аналогичный вариант вызова метода `prop()`. Различие между обоими методами состоит в том, что методы `prop()` имеют дело со свойствами, определяемыми объектами `HTMLElement`, а не со значениями атрибутов. Часто атрибуты и соответствующие свойства имеют одинаковые имена, но это не всегда так. В качестве простого примера можно привести атрибут `class`, представленный в объекте `HTMLElement` свойством `className`. Пример использования метода `prop()` для получения значения этого свойства приведен в листинге 8.7.

Листинг 8.7. Использование метода `prop()` для получения значения свойства

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('*[class]').each(function(index, elem) {
            console.log("Элемент:" + elem.tagName +
                " " + $(elem).prop("className"));
        });
    });
</script>
...
```

В этом примере сначала выбираются в виде набора, а затем последовательно перебираются с помощью метода `each()` все элементы, у которых имеется атрибут `class`. Для каждого элемента на консоль выводится его тип и значение свойства `className`.

Работа с классами

Несмотря на то что для работы с классами можно использовать общие методы, предназначенные для работы с атрибутами, в jQuery предусмотрены намного более удобные методы, специально предназначенные для этой цели. Их краткое описание приведено в табл. 8.3.

Таблица 8.3. Методы для работы с классами

Метод	*Описание*
`addClass(`*имя имя ...*`)`	Добавляет один или несколько классов с указанными именами во все элементы, содержащиеся в объекте `jQuery`
`addClass(`*функция*`)`	Добавляет классы, список которых возвращает указанная функция, во все элементы, содержащиеся в объекте `jQuery`
`hasClass(`*имя*`)`	Возвращает `true`, если хотя бы один из элементов, содержащихся в объекте `jQuery`, принадлежит указанному классу
`removeClass(`*имя имя ...*`)`	Удаляет один или несколько классов с указанными именами из всех элементов, содержащихся в объекте `jQuery`
`removeClass(`*функция*`)`	Удаляет классы, список которых возвращает указанная функция, из всех элементов, содержащихся в объекте `jQuery`

Окончание табл. 8.3

Метод	Описание
`toggleClass()`	Переключает все классы, которым принадлежат элементы, содержащиеся в объекте `jQuery`
`toggleClass(`*логическое_значение*`)`	Осуществляет одностороннее переключение всех классов, которым принадлежат элементы, содержащиеся в объекте `jQuery`
`toggleClass(`*имя*`)` `toggleClass(`*имя имя ...*`)`	Переключает один или несколько классов с указанными именами для всех элементов, содержащихся в объекте `jQuery`
`toggleClass(`*имя*`,` *логическое_значение*`)`	Осуществляет одностороннее переключение класса с указанным именем для всех элементов, содержащихся в объекте `jQuery`
`toggleClass(`*функция*`,` *логическое_значение*`)`	Динамически переключает классы для всех элементов, содержащихся в объекте `jQuery`

Можно добавлять классы в элементы, используя метод `addClass()`, или удалять классы, используя метод `removeClass()`, а также проверять принадлежность элемента определенному классу, используя метод `hasClass()`. Пример использования этих методов приведен в листинге 8.8.

Листинг 8.8. Добавление и удаление классов, а также проверка их наличия

```
...
<style type="text/css">
    img.redBorder {border: thick solid red}
    img.blueBorder {border: thick solid blue}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').addClass("redBorder");
        $('img:even').removeClass("redBorder")
            .addClass("blueBorder");

        console.log("Все элементы: " + $('img')
            .hasClass('redBorder'));
        $('img').each(function(index, elem) {
            console.log("Элемент: " + $(elem)
                .hasClass('redBorder') + " " + elem.src);
        });

    });
</script>
...
```

В начале этого фрагмента кода помещен элемент `style`, содержащий определения двух стилей, применение которых зависит от принадлежности элемента определенному классу. Привлекать классы для управления CSS вовсе не обязательно, однако их использование позволяет упростить демонстрацию эффектов изменений, изучаемых в данной главе.

Сценарий начинается с выбора всех элементов `img` в документе и присвоения им класса `redBorder` с помощью метода `addClass()`. Затем мы выбираем четные элементы `img`, удаляем их из класса `redBorder` и присваиваем им класс `blueBorder`, используя метод `removeClass()`.

Совет. Метод `addClass()` не удаляет существующие классы из элементов, а лишь добавляет новые классы в дополнение к тем, которые были применены к элементам ранее.

Наконец, мы используем метод `hasClass()` для проверки на принадлежность классу `redBorder` как всего выбранного набора в целом (этот метод возвращает `true`, если хотя бы один элемент набора имеет указанный класс), так и каждого элемента `img` по отдельности. Вид страницы в окне браузера приведен на рис. 8.5.

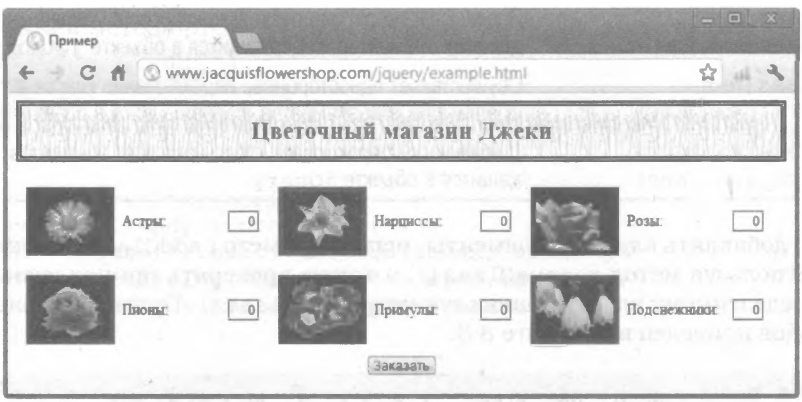


Рис. 8.5. Применение стилей в зависимости от принадлежности элементов определенным классам

При этом на консоль выводится следующий результат.

```
Все элементы: true
Элемент: false http://www.jacquisflowershop.com/jquery/astor.png
Элемент: true http://www.jacquisflowershop.com/jquery/daffodil.png
Элемент: false http://www.jacquisflowershop.com/jquery/rose.png
Элемент: true http://www.jacquisflowershop.com/jquery/peony.png
Элемент: false http://www.jacquisflowershop.com/jquery/primula.png
Элемент: true http://www.jacquisflowershop.com/jquery/snowdrop.png
```

Добавление и удаление классов с помощью функции

Классы можно динамически добавлять или удалять из набора элементов, передавая методам `addClass()` и `removeClass()` функцию в качестве аргумента. Пример использования функции совместно с методом `addClass()` приведен в листинге 8.9.

Листинг 8.9. Использование функции в методе `addClass()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img').addClass(function(index, currentClasses) {
            if (index % 2 == 0) {
                return "blueBorder";
            } else {
                return "redBorder";
            }
```

```
        });
    });
</script>
...
```

Аргументами функции являются индекс элемента в наборе выбранных элементов, а также текущий набор классов, членом которых является данный элемент. Как и в случае других аналогичных функций, переменная `this` ссылается на объект `HTMLElement` текущего обрабатываемого элемента. Функция должна возвращать класс, к которому требуется присоединить элемент. В данном примере решение о том, какой именно из классов следует назначить элементу — `redBorder` или `blueBorder`, — принимается на основании значения аргумента `index`. Вид страницы в окне браузера будет тем же, что и на рис. 8.5.

Аналогичный подход применяется и при удалении классов из элементов. Для этого следует передать методу `removeClass()` соответствующую функцию, как показано в листинге 8.10.

Листинг 8.10. Удаление классов из элементов с помощью функции

```
...
<style type="text/css">
    img.redBorder {border: thick solid red}
    img.blueBorder {border: thick solid blue}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");

        $('img').removeClass(function(index, currentClasses) {
            if ($(this).closest('#row2').length > 0
                && currentClasses.indexOf('redBorder') > -1) {
                    return "redBorder";
            } else {
                return "";
            }
        });
    });
</script>
...
```

В этом сценарии функция, передаваемая методу `removeClass()`, использует объект `HTMLElement` и текущий набор классов для удаления класса `redBorder` из любого элемента, который принадлежит этому классу и является потомком элемента с `id`, равным `row2`. Вид страницы в окне браузера приведен на рис. 8.6.

Совет. Обратите внимание, что в тех случаях, когда классы не должны удаляться, функция должна возвращать пустую строку. В случае отсутствия возвращаемого значения jQuery удаляет из элемента *все* классы.

Переключение отдельного класса

В своей простейшей форме переключение класса означает добавление указанного класса, если элемент не имеет данного класса, или удаление класса, если эле-

мент имеет данный класс. Это достигается передачей методу `toggleClass()` имени класса, как показано в листинге 8.11.

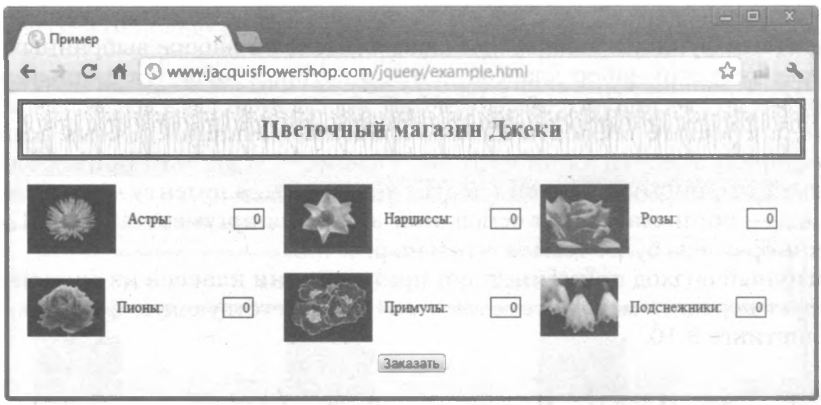


Рис. 8.6. Удаление классов с помощью функции

Листинг 8.11. Использование метода `toggleClass()`

```
...
<style type="text/css">
    img.redBorder {border: thick solid red}
    img.blueBorder {border: thick solid blue}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");

        $("<button>Переключить</button>")
            .appendTo("#buttonDiv").click(doToggle);

        function doToggle(e) {
            $('img').toggleClass("redBorder");
            e.preventDefault();
        };

    });
</script>
...
```

Этот сценарий начинается с присвоения класса `redBorder` нечетным элементам `img`, а класса `blueBorder` — четным. Затем мы создаем элемент `button` и присоединяем его к элементу с атрибутом `id`, равным `buttonDiv`. В результате этого новая кнопка располагается рядом с уже имеющейся на странице кнопкой Заказать. Функция, которую jQuery должна вызывать после щелчка на кнопке, задается с помощью метода `click()`. Эта возможность обеспечивается предоставляемой в jQuery поддержкой событий, о чем говорится в главе 9.

Ключевая инструкция функции `doToggle()`, которая вызывается после щелчка на кнопке, выглядит так:

```
$('img').toggleClass("redBorder");
```

Она выбирает все элементы `img` в документе и управляет переключением класса `redBorder`. Аргумент функции `doToggle()` и вызов метода `preventDefault()` интереса для нас пока что не представляют. Речь о них пойдет в главе 9. Результаты работы данного сценария вы можете увидеть на рис. 8.7, однако будет лучше, если вы загрузите данный документ в браузер и сами выполните щелчок на кнопке.

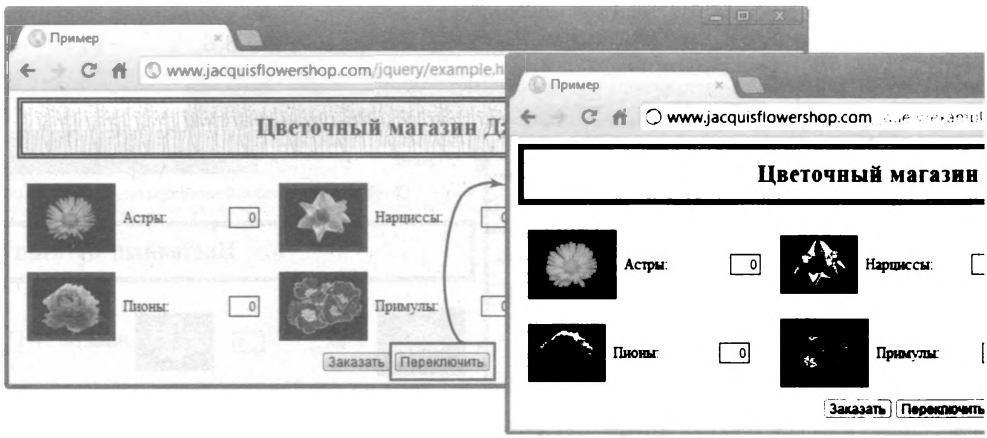


Рис. 8.7. Переключение принадлежности классу с помощью метода `toggleClass`

Если вы наблюдательны, то должны были заметить в этом рисунке одну особенность. Дело в том, что у элементов, заключенных в красную рамку, рамка исчезает, а элементы, которые были первоначально заключены в синюю рамку, таковыми и остаются. Суть происходящего состоит в том, что jQuery, как и следовало ожидать, удаляет класс `redBorder` из нечетных элементов `img` и добавляет его в четные элементы, но элемент, к которому добавляется класс `redBorder`, является также членом класса `blueBorder`. Стиль `blueBorder` определен в элементе `style` после стиля `redBorder`, откуда следует, что значения его свойств имеют более высокий приоритет, как объяснялось в главе 3. Поэтому переключение классов работает, но при этом должны учитываться также некоторые тонкости функционирования CSS. Если вы хотите, чтобы красные рамки проявлялись в обоих случаях, следует обратить порядок объявления стилей, как показано в листинге 8.12.

Листинг 8.12. Согласование порядка объявления стилей с переключением классов

```
...
<style type="text/css">
    img.blueBorder {border: thick solid blue}
    img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
    $(document).ready(function() {
        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");

        $("<button>Переключить</button>")
            .appendTo("#buttonDiv").click(doToggle);

        function doToggle(e) {
            $('img').toggleClass("redBorder");
```

```
            e.preventDefault();
        };
    });
</script>
...
```

Теперь, если элемент принадлежит одновременно классам `blueBorder` и `redBorder`, браузер будет использовать значение свойства `border`, определенное для класса `redBorder`. Результаты внесения изменений отражены на рис. 8.8.

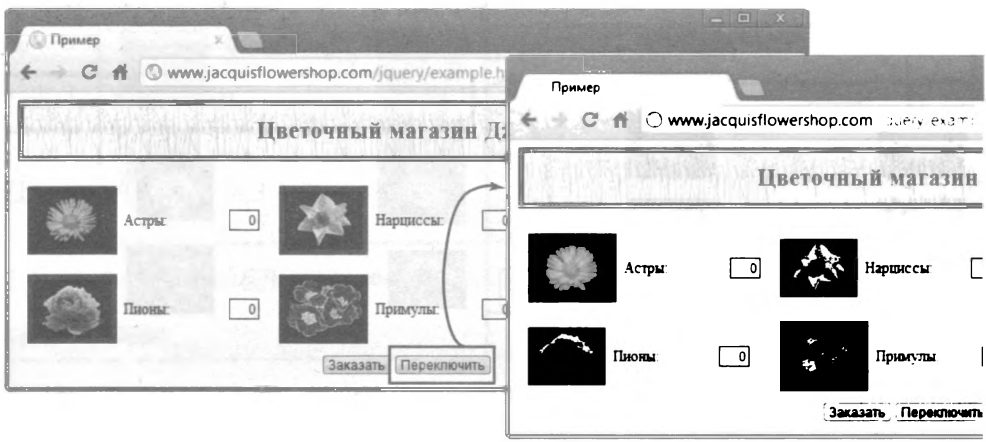


Рис. 8.8. Эффект согласования порядка объявления стилей в CSS с переключением классов

Переключение одновременно нескольких классов

Если методу `toggleClass` передается список имен классов, разделенных пробелами, то соответствующее переключение будет выполнено для каждого из элементов выбранного набора. Один их примеров того, как можно сделать, приведен в листинге 8.13.

Листинг 8.13. Переключение группы классов

```
...
<style type="text/css">
    img.blueBorder {border: thick solid blue}
    img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");

        $("<button>Переключить</button>")
            .appendTo("#buttonDiv").click(doToggle);

        function doToggle(e) {
            $('img').toggleClass("redBorder blueBorder");
            e.preventDefault();
        };
```

```
    });
</script>
...
```

В этом примере для всех элементов `img` осуществляется переключение классов `redBorder` и `blueBorder`. Результат представлен на рис. 8.9.

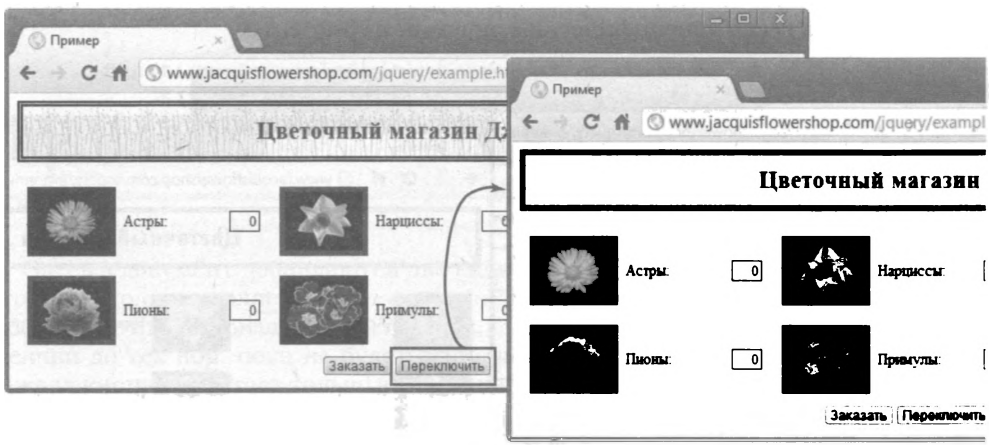


Рис. 8.9. Одновременное переключение группы классов

Переключение всех классов

Можно переключить сразу все классы, которым принадлежит набор элементов, вызвав метод `toggleClass()` без аргументов. Это более "умный" прием, поскольку jQuery сохраняет информацию о переключаемых классах, так что они присваиваются и удаляются корректно. Пример использования этой методики приведен в листинге 8.14.

Листинг 8.14. Переключение всех классов для выбранных элементов

```
...
<style type="text/css">
    img.blueBorder {border: thick solid blue}
    img.redBorder {border: thick solid red}
    label.bigfont {font-size: 1.5em}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");
            $('label').addClass("bigFont");

        $("<button>Переключить</button>")
            .appendTo("#buttonDiv").click(doToggle);

        function doToggle(e) {
            $('img, label').toggleClass();
            e.preventDefault();
        };
```

```
    });
</script>
...
```

В этом примере метод `addClass()` используется для добавления классов во все элементы `img` и `label`. После щелчка на кнопке Переключить осуществляется выбор указанных элементов и вызывается метод `toggleClass()` без аргументов. Результат, являющийся весьма специфическим, показан на рис. 8.10.

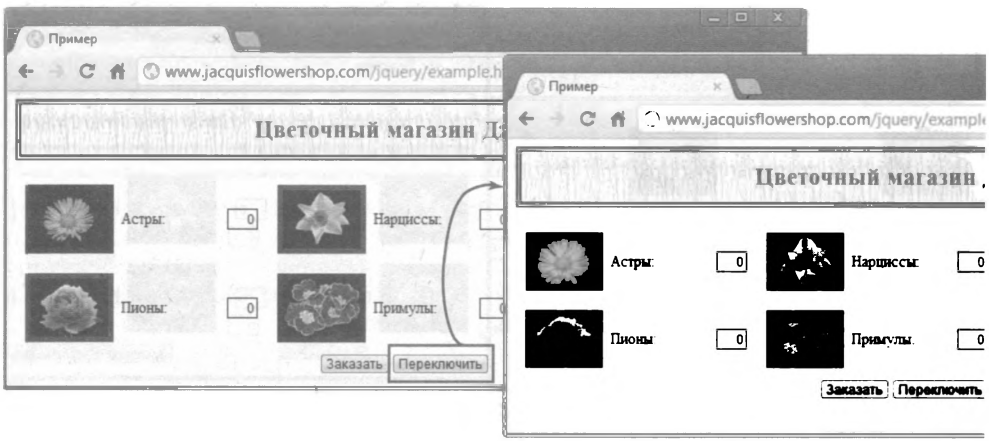


Рис. 8.10. Переключение всех классов для элементов

После первого щелчка на кнопке из выделенных элементов удаляются все классы, однако jQuery запоминает, какие классы были удалены, и поэтому может присвоить их элементам заново после выполнения повторного щелчка.

Одностороннее переключение классов

Процессом переключения классов (т.е. их добавлением, если они есть, или удалением, если их нет) можно управлять, передавая методу `toggleClass()` логическое (булево) значение в качестве аргумента. Если передается значение `false`, то классы будут только удаляться, если `true` — только добавляться. Того же эффекта можно добиться совместным использованием методов `addClass()` и `removeClass()`, и поэтому данной возможностью я пользуюсь относительно редко. Соответствующий пример приведен в листинге 8.15.

Листинг 8.15. Одностороннее переключение классов

```
...
<style type="text/css">
    img.blueBorder {border: thick solid blue}
    img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");

        $("<button>Переключить</button>")
```

```
            .appendTo("#buttonDiv").click(doToggleOn);
        $("<button>Вернуть</button>")
            .appendTo("#buttonDiv").click(doToggleOff);

        function doToggleOff(e) {
            $('img, label').toggleClass("redBorder", false);
            e.preventDefault();
        };
        function doToggleOn(e) {
            $('img, label').toggleClass("redBorder", true);
            e.preventDefault();
        };
    });
</script>
...
```

Здесь в документ добавляются две кнопки, одна из которых только добавляет, а вторая — только удаляет класс `redBorder` из выбранных элементов. Как только на любой из этих кнопок выполнен щелчок, она не будет реагировать на последующие щелчки до тех пор, пока не будет выполнен щелчок на другой кнопке (поскольку каждая кнопка работает только в одном направлении).

Динамическое переключение классов

Решение о том, на какие классы должно воздействовать переключение, может приниматься "на лету" за счет передачи методу `toggleClass()` функции обратного вызова. Простой пример того, как это можно сделать, приведен в листинге 8.16.

Листинг 8.16. Переключение классов с помощью функции

```
...
<style type="text/css">
    img.blueBorder {border: thick solid blue}
    img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').addClass("blueBorder");
        $('img:even').addClass("redBorder");

        $("<button>Переключить</button>")
            .appendTo("#buttonDiv").click(doToggle);

        function doToggle(e) {
            $('img').toggleClass(function(index, currentClasses) {
                if (index % 2 == 0) {
                   return "redBorder";
                } else {
                    return "";
                }
            });
            e.preventDefault();
        };
    });
</script>
...
```

Здесь класс `blueBorder` применяется ко всем элементам `img`, а класс `redBorder` — только к четным. Аргументами, которые передаются функции, являются индекс текущего обрабатываемого элемента и текущий набор классов, которым он принадлежит. Переменная `this` ссылается на объект `HTMLElement` текущего элемента. Результатом, который возвращает функция, является список имен классов, переключение которых следует выполнить. Если переключение классов не требуется, функция должна возвратить пустую строку (отсутствие возвращаемого функцией результата для какого-либо элемента приводит к переключению всех его классов).

Работа с CSS

В ряде предыдущих примеров для установки значения атрибута `style` и, следовательно, для установки значений CSS-свойств элементов использовались базовые методы, предназначенные для работы с атрибутами. Библиотека jQuery предоставляет ряд удобных специализированных методов, значительно упрощающих работу со стилями CSS. Одним из наиболее широко используемых методов такого рода является метод `css()`, краткое описание которого приведено в табл. 8.4.

Таблица 8.4. Метод `css()`

Метод	*Описание*
`css(`*имя*`)`	Возвращает значение указанного свойства для первого из элементов, содержащихся в объекте `jQuery`
`css(`*имя, значение*`)`	Устанавливает значение указанного свойства для всех элементов, содержащихся в объекте `jQuery`
`css(`*отображение*`)`	Устанавливает одновременно несколько свойств для всех элементов, содержащихся в объекте `jQuery`, с помощью объекта отображения
`css(`*имя, функция*`)`	Устанавливает значения указанного свойства для всех элементов, содержащихся в объекте `jQuery`, с помощью функции

При считывании значений свойств с помощью метода `css()` вы получаете значение свойства, которое имеет первый из элементов, содержащихся в объекте `jQuery`. В то же время при установке свойства вносимое изменение применяется ко всем элементам набора. Пример простейшего использования метода `css()` приведен в листинге 8.17.

Листинг 8.17. Использование метода `css()` для получения и установки значений свойств CSS

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var sizeVal = $('label').css("font-size");
        console.log("Размер: " + sizeVal);
        $('label').css("font-size", "1.5em");
    });
</script>
...
```

В этом сценарии мы выбираем все элементы `label`, получаем с помощью метода `css()` значение свойства `font-size` и выводим его на консоль. Затем мы вновь вы-

бираем все элементы label и присваиваем новое значение этого же свойства всем элементам.

Совет. Несмотря на то что в сценарии используется фактическое имя свойства (font-size), а не его запись с применением "ВерблюжьегоРегистра", т.е. форма записи, в которой это свойство определено в объекте HTMLElement (свойство fontSize), оно также воспринимается корректно, поскольку jQuery поддерживает оба варианта.

В результате работы данного сценария на консоль выводится следующий результат.

```
Размер: 16px
```

Совет. Установка для свойства значения в виде пустой строки ("") равносильна удалению этого свойства из атрибута style данного элемента.

Установка одновременно нескольких свойств CSS

Существуют два способа одновременной установки нескольких CSS-свойств. Первый из них — это формирование цепочки вызовов метода css(), как показано в листинге 8.18.

Листинг 8.18. Цепочка вызовов метода css()

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('label').css("font-size", "1.5em").css("color", "blue");
    });
</script>
...
```

В этом сценарии устанавливаются значения свойств font-size и color. Того же эффекта можно добиться, используя объект отображения, как показано в листинге 8.19.

Листинг 8.19. Одновременная установка нескольких свойств с помощью объекта отображения

```
...
<script type="text/javascript">
  $(document).ready(function() {
      var cssVals = {
          "font-size": "1.5em",
          "color": "blue"
      };

      $('label').css(cssVals);
  });
</script>
...
```

Оба варианта сценария приводят к одному и тому же результату, показанному на рис. 8.11.

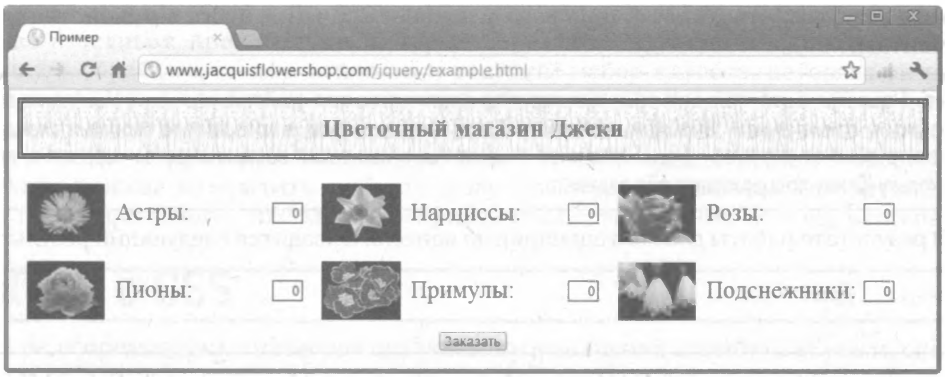


Рис. 8.11. Одновременная установка нескольких свойств

Установка относительных значений

В качестве аргументов метод `css()` может принимать также относительные значения. Они представляют собой числовые значения, которым предшествуют знаки `+=` и `-=` и которые добавляются или вычитаются из текущего значения. Этот прием можно использовать лишь в отношении числовых величин. Соответствующий пример приведен в листинге 8.20.

Листинг 8.20. Использование относительных значений в методе `css()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('label:odd').css("font-size", "+=5")
        $('label:even').css("font-size", "-=5")

    });
</script>
...
```

Относительные значения выражаются в тех же единицах, что и значения свойств. В данном случае размер шрифта увеличивается на 5 пикселей для нечетных элементов `label` и уменьшается на ту же величину для четных. Результат показан на рис. 8.12.

Установка свойств с помощью функции

Можно устанавливать значения свойств динамически, передавая функцию методу `css()`. Соответствующий пример приведен в листинге 8.21. Аргументами, передаваемыми функции, являются индекс элемента в наборе и текущее значение свойства. Переменная `this` ссылается на объект `HTMLElement`, соответствующий элементу, а функция возвращает значение, которое требуется установить.

Листинг 8.21. Установка CSS-значений с помощью функции

```
...
<script type="text/javascript">
```

```
    $(document).ready(function() {
        $('label').css("border", function(index, currentValue) {
            if ($(this).closest("#row1").length > 0) {
                return "thick solid red";
            } else if (index % 2 == 1) {
                return "thick double blue";
            }
        });
    });
</script>
...
```

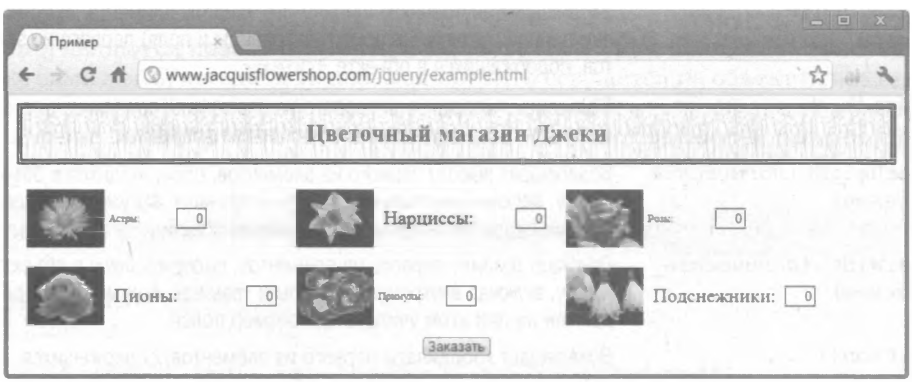


Рис. 8.12. Использование относительных значений

Результат работы этого сценария представлен на рис. 8.13.

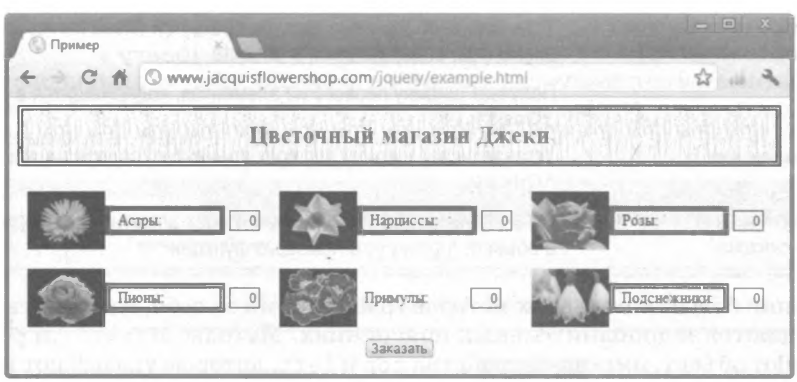


Рис. 8.13. Установка значений свойств CSS с помощью функции

Использование специализированных методов для работы со свойствами CSS

В дополнение к методу `css()` в jQuery определен ряд специализированных методов, предназначенных для получения или установки значений конкретных свойств. Перечень этих методов приведен в табл. 8.5.

Таблица 8.5. Методы для работы с конкретными свойствами CSS

Метод	*Описание*
`height()`	Возвращает высоту (в пикселях) первого из элементов, содержащихся в объекте `jQuery`
`height(`*значение*`)`	Устанавливает высоту для всех элементов, содержащихся в объекте `jQuery`
`innerHeight()`	Возвращает внутреннюю высоту (т.е. высоту элемента, включая внутренние отступы, но исключая границы и поля) первого из элементов, содержащихся в объекте `jQuery`
`innerWidth()`	Возвращает внутреннюю ширину (т.е. ширину элемента, включая внутренние отступы, но исключая границы и поля) первого из элементов, содержащихся в объекте `jQuery`
`offset()`	Возвращает координаты первого из элементов, содержащихся в объекте `jQuery`, относительно начала документа
`outerHeight(`*логическое_значение*`)`	Возвращает высоту первого из элементов, содержащихся в объекте jQuery, включая внутренние отступы и границы. Аргумент определяет, должен ли при этом учитываться размер полей
`outerWidth(`*логическое_значение*`)`	Получает ширину первого из элементов, содержащихся в объекте jQuery, включая внутренние отступы и границы. Аргумент определяет, должен ли при этом учитываться размер полей
`position()`	Возвращает координаты первого из элементов, содержащихся в объекте `jQuery`, относительно его родительского элемента, у которого задан тип позиционирования
`scrollLeft()` `scrollTop()`	Получает значение отступа прокрутки слева или сверху для первого из элементов, содержащихся в объекте `jQuery`
`scrollLeft(`*значение*`)` `scrollTop(`*значение*`)`	Устанавливает значение отступа прокрутки слева или сверху для всех элементов, содержащихся в объекте `jQuery`
`width()`	Получает ширину первого из элементов, содержащихся в объекте `jQuery`
`width(`*значение*`)`	Устанавливает ширину для всех элементов, содержащихся в объекте `jQuery`
`height(`*функция*`)` `width(`*функция*`)`	Устанавливает высоту или ширину всех элементов, содержащихся в объекте `jQuery`, с помощью функции

Названия большинства этих методов говорят сами за себя, однако некоторые из них нуждаются в дополнительных пояснениях. Методы `offset()` и `position()` возвращают объект, имеющий свойства `top` и `left`, которые указывают положение элемента. Пример использования метода `position()` приведен в листинге 8.22.

Листинг 8.22. Использование метода `position()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var pos = $('img').position();
        console.log("Position top: " + pos.top +
            " left: " + pos.left);
```

```
    });
</script>
...
```

Этот сценарий выводит на консоль значения свойств `top` и `left` объекта, возвращаемого данным методом. Результат выглядит следующим образом.

```
Положение top: 108 left: 18
```

Установка ширины и высоты с помощью функции

Ширину и высоту элементов набора можно устанавливать динамически, передавая методам `width()` и `height()` функцию в качестве аргумента. Аргументами функции являются позиция элемента в наборе и текущее значение свойства. Как вы уже сами могли догадаться, переменная `this` ссылается на объект `HTMLElement`, соответствующий текущему обрабатываемому элементу, а функция возвращает значение, которое требуется установить. Соответствующий пример приведен в листинге 8.23.

Листинг 8.23. Установка высоты элементов с помощью функции

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#row1 img').css("border", "thick solid red")
            .height(function(index, currentValue) {
                return (index + 1) * 25;
            });
    });
</script>
...
```

В этом сценарии значение индекса, определяющего позицию элемента в наборе, используется в качестве множителя, регулирующего высоту элемента. Результат представлен на рис. 8.14.

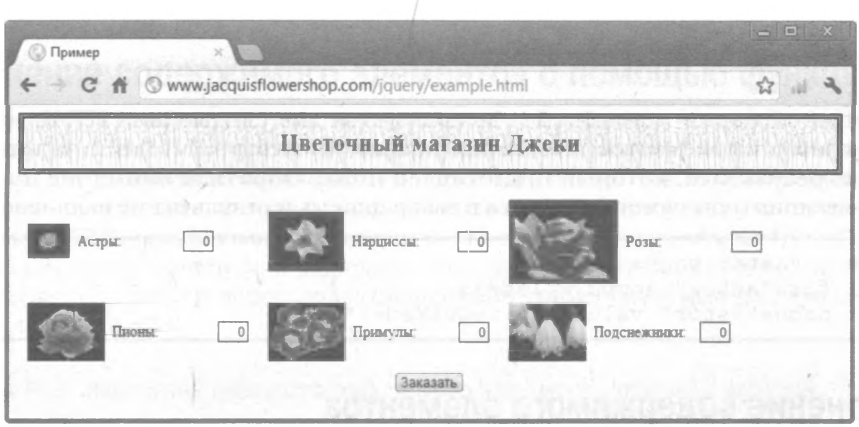


Рис. 8.14. Использование функции для установки высоты элементов

Работа с содержимым элементов

До сих пор мы занимались поиском атрибутов элементов, но в jQuery предусмотрены средства, позволяющие работать также с содержимым элементов. Доступные для этого методы приведены в табл. 8.6.

Таблица 8.6. Методы для работы с содержимым элементов

Метод	*Описание*
`text()`	Получает объединенное текстовое содержимое всех элементов, содержащихся в объекте `jQuery`, и их потомков
`text(`*значение*`)`	Устанавливает текстовое содержимое для всех элементов, содержащихся в объекте `jQuery`
`html()`	Возвращает HTML-содержимое первого из элементов, содержащихся в объекте `jQuery`
`html(`*значение*`)`	Устанавливает HTML-содержимое для всех элементов, содержащихся в объекте `jQuery`
`text(`*функция*`)` `html(`*функция*`)`	Устанавливает текстовое и HTML-содержимое с помощью функции

С методом `text()` связана одна особенность: если он вызывается без аргументов, то возвращаемый им результат генерируется на основе содержимого всех выбранных элементов. В отличие от этого поведение метода `html()` согласуется с поведением остальных методов jQuery, и он возвращает лишь содержимое первого элемента выбранного набора, как показано в листинге 8.24.

Листинг 8.24. Использование метода `html()` для считывания содержимого элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var html = $('div.dcell').html();
        console.log(html);
    });
</script>
...
```

В этом сценарии метод `html()` используется для считывания HTML-содержимого первого из элементов, соответствующих селектору `div.dcell`, и вывода на консоль результата, который представлен ниже. Обратите внимание на то, что HTML-дескрипторы самого элемента в возвращаемый результат не включаются.

```
<img src="astor.png">
<label for="astor">Астры:</label>
<input name="astor" value="0" required="">
```

Изменение содержимого элементов

Для изменения содержимого элементов используются методы `html()` и `text()`. В нашем примере документа для цветочного интернет-магазина сколь-нибудь зна-

чимое текстовое содержимое элементов отсутствует, поэтому в листинге 8.25 показано лишь применение метода `html()` для этой цели.

Листинг 8.25. Использование метода `html()` для изменения содержимого элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#row2 div.dcell').html($('div.dcell').html());
    });
</script>
...
```

Этот сценарий изменяет HTML-содержимое тех принадлежащих классу `dcell` элементов `div`, которые являются потомками элемента `row2`. Для получения содержимого, подлежащего вставке, используется метод `html()`, который считывает HTML-содержимое из первого элемента `div.dcell`. Это приводит к тому, что каждое из изображений в нижнем ряду элементов заменяется изображением астры, как показано на рис. 8.15.

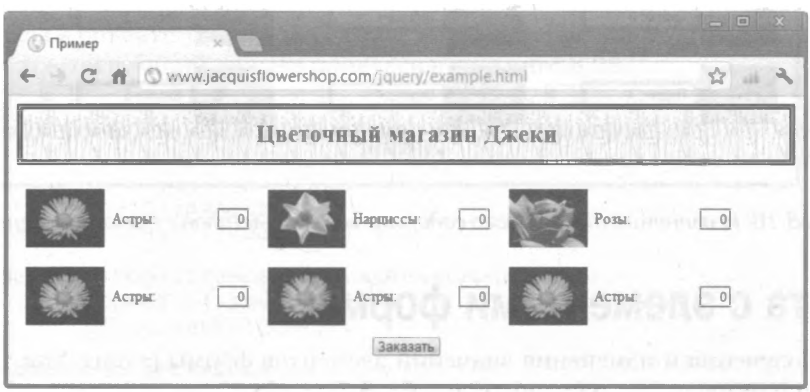


Рис. 8.15. Изменение содержимого элементов с помощью метода `html()`

Изменение содержимого элементов с помощью функции

Как и в случае многих других методов, рассматриваемых в данной главе, передача методам `html()` и `text()` функции в качестве аргумента обеспечивает возможность динамического изменения содержимого элементов. В обоих случаях аргументами функции являются позиция элемента в объекте `jQuery` и текущее текстовое или HTML-содержимое. Переменная `this` ссылается на объект `HTMLElement`, соответствующий элементу, а возвращаемое значение функции содержит требуемый результат. Пример использования функции совместно с методом `text()` приведен в листинге 8.26.

Листинг 8.26. Изменение текстового содержимого элементов с помощью функции

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $('label').css("border", "thick solid red")
            .text(function(index, currentValue) {
            return "Индекс " + index;
        });
    });
</script>
...
```

В этом сценарии текстовое содержимое элементов `label` изменяется в зависимости от значения аргумента `index`, указывающего позицию элемента в выбранном наборе (для заключения изменяемых элементов в рамку используется метод `css()`). Результат представлен на рис. 8.16.

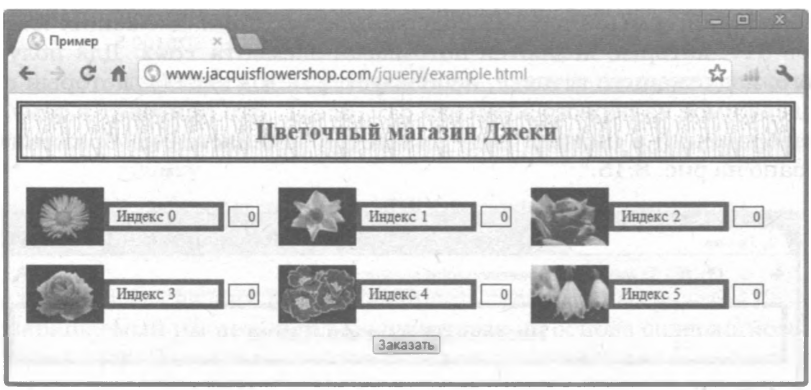


Рис. 8.16. Изменение текстового содержимого элементов с помощью функции

Работа с элементами формы

Для получения и изменения значений элементов формы (таких, как `input`) используется метод `val()`, описанный в табл. 8.7.

Таблица 8.7. Метод `val()`

Метод	Описание
`val()`	Возвращает значение первого из элементов, содержащихся в объекте `jQuery`
`val(`*значение*`)`	Изменяет значения всех элементов, содержащихся в объекте `jQuery`
`val(`*функция*`)`	Изменяет значения всех элементов, содержащихся в объекте `jQuery`, с помощью функции

Пример использования метода `val()` для получения значения первого из элементов, содержащихся в объекте `jQuery`, приведен в листинге 8.27. Метод `each()` используется для перебора значений всех элементов `input`, содержащихся в документе.

Листинг 8.27. Использование метода `val()` для получения значений элементов `input`

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $('input').each(function(index, elem) {
            console.log("Имя: " + elem.name +
                " Значение: " + $(elem).val());
        });
    });
    </script>
...
```

На консоль выводится следующий результат.

```
Имя: astor Значение: 0
Имя: daffodil Значение: 0
Имя: rose Значение: 0
Имя: peony Значение: 0
Имя: primula Значение: 0
Имя: snowdrop Значение: 0
```

Изменение значений элементов формы

Метод `val()` можно использовать для изменения значений всех элементов, содержащихся в объекте `jQuery`, передавая ему требуемое значение в качестве аргумента. Соответствующий пример приведен в листинге 8.28.

Листинг 8.28. Изменение значений элементов с помощью метода `val()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Установить значения</button>")
            .appendTo("#buttonDiv")
            .click(setValues);

        function setValues(e) {
            $('input').val(100);
            e.preventDefault();
        }
    });
</script>
...
```

В этом сценарии в документ добавляется кнопка, после щелчка на которой вызывается функция `setValues()`. Эта функция выбирает все элементы `input` в документе и устанавливает их значения равными `100` с помощью метода `val()`. Результат представлен на рис. 8.17.

Изменение значений элементов формы с помощью функции

Как несложно догадаться, для изменения значений с помощью метода `val()` можно использовать также функцию. Аргументами этой функции являются позиция элемента в наборе и его текущее значение. Переменная `this` ссылается на объект `HTMLElement`, представляющий обрабатываемый элемент. Используя этот способ, можно устанавливать значения элементов динамически, как показано в листинге 8.29.

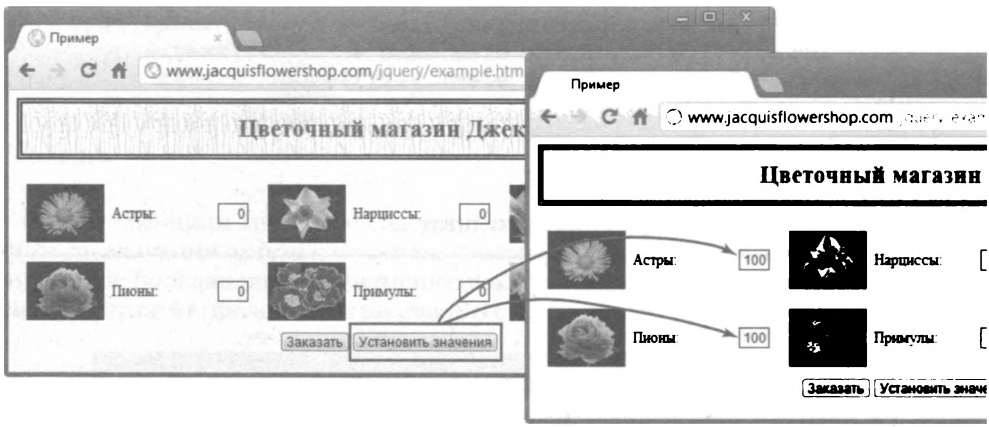


Рис. 8.17. Использование метода `val()` для изменения значений элементов `input`

Листинг 8.29. Использование метода `val()` совместно с функцией

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('input').val(function(index, currentVal) {
            return (index + 1) * 100;
        });
    });
</script>
...
```

В этом примере значения элементов устанавливаются на основании значения аргумента `index`, указывающего позицию элемента в выбранном наборе. Результат представлен на рис. 8.18.

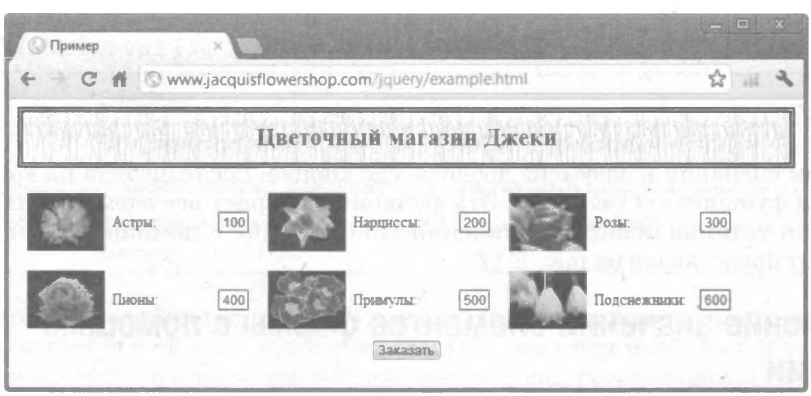


Рис. 8.18. Динамическое изменение значений элементов путем использования метода `val()` совместно с функцией

Связывание данных с элементами

Библиотека jQuery позволяет связывать с элементами произвольные данные, которые впоследствии можно извлекать и тестировать. Перечень предназначенных для этого методов содержится в табл. 8.8.

Таблица 8.8. Методы для работы с произвольными данными элементов

Метод	*Описание*
`data(`*ключ, значение*`)` `data(`*отображение*`)`	Связывает одну или несколько пар "ключ-значение" с элементами, содержащимися в объекте `jQuery`
`data(`*ключ*`)`	Возвращает значение указанного ключа, связанное с первым из элементов, содержащихся в объекте `jQuery`
`data()`	Возвращает пары "ключ-значение", связанные с первым из элементов, содержащихся в объекте `jQuery`
`removeData(`*ключ*`)`	Удаляет данные, связанные с этим ключом, из всех элементов, содержащихся в объекте `jQuery`
`removeData()`	Удаляет все элементы данных из всех элементов, содержащихся в объекте `jQuery`

Пример установки, тестирования, получения и удаления данных, связанных с элементами, приведен в листинге 8.30.

Примечание. В случае использования метода `clone()` связанные с элементами данные удаляются из вновь скопированных элементов, если только вы не указали jQuery в явной форме на необходимость их сохранения. Более подробно о методе `clone()` и о том, как сохраняются данные, рассказано в главе 7.

Листинг 8.30. Обработка данных, связанных с элементами

```
...
<script type="text/javascript">
    $(document).ready(function() {

        // установить данные
        $('img').each(function () {
           $(this).data("product", $(this)
                   .siblings("input[name]")
                   .attr("name"));
        });

        // найти элементы с данными и получить значения
        $('*').filter(function() {
            return $(this).data("product") != null;
        }).each(function() {
           console.log("Элемент: " + this.tagName + " " +
               $(this).data("product"));
        });

        // удалить все данные
        $('img').removeData();
```

```
    });
</script>
...
```

В этом сценарии можно выделить три отдельных этапа выполнения. На первом этапе метод `data()` используется для связывания элемента данных с ключом `product`. Данные получаются путем перехода от каждого элемента `img` к его сестринскому элементу `input` с атрибутом `name`.

На втором этапе сначала выбираются все элементы в документе, а затем с помощью метода `filter()` находятся те из них, у которых имеется значение, связанное с ключом `product`. Далее осуществляется перебор этих элементов с помощью метода `each()` и вывод данных на консоль. Здесь бросается в глаза, что операции дублируются, но я хотел продемонстрировать наилучшую методику выбора элементов, с которыми связаны данные. Специально предназначенных для этого селекторов или методов не существует, поэтому мы решаем данную задачу путем использования метода `filter()` совместно с функцией.

Наконец, метод `removeData()` используется для удаления всех данных из всех элементов `img`. Консольный вывод выглядит следующим образом.

```
Элемент: IMG astor
Элемент: IMG daffodil
Элемент: IMG rose
Элемент: IMG peony
Элемент: IMG primula
Элемент: IMG snowdrop
```

Работа с атрибутами данных HTML5

В спецификации HTML5 определяются *атрибуты данных* (data attributes), которые также позволяют связывать данные элементов. Атрибутам данных, называемым также *расширенными атрибутами* (expand attributes), присваивают имена с префиксом `data`, и их очень удобно использовать для придания элементам дополнительного смысла, что не всегда удается обеспечить в полной мере с помощью классов. Метод `data()` позволяет автоматически получать и изменять атрибуты данных, как показано в листинге 8.31.

Листинг 8.31. Использование метода `data()` совместно с атрибутами данных

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('div.dcell').each(function () {
                var productVal = $(this).data("product");
                console.log("Продукт: " + productVal);
            });
        });
    </script>
</head>
```

```
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell" data-product="astor">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell" data-product="daffodil">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell" data-product="rose">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button></div>
    </form>
</body>
</html>
```

В этом примере атрибуты данных HTML5 добавляются в усеченную версию нашего образца документа. В сценарии сначала выбираются элементы с атрибутами данных, а затем связанные с элементами данные извлекаются с помощью метода `data()` и выводятся на консоль. Заметьте, что в имени атрибута префикс `data-` опущен — в качестве ссылки на атрибут `data-product` в сценарии используется просто `product`. На консоль выводится следующий результат.

```
Продукт: astor
Продукт: daffodil
Продукт: rose
```

Совет. Метод `data()` учитывает атрибуты данных и при установке значений. Если вы указываете некоторый ключ, например `product`, то метод `data()` проверяет, существует ли соответствующий атрибут данных HTML5, например `data-product`. Если это так, то указанное вами значение присваивается атрибуту. В противном случае данные сохраняются внутри jQuery.

Резюме

В данной главе вы познакомились со способами манипулирования элементами DOM. Здесь было показано, как получать и устанавливать значения атрибутов, в том числе с помощью вспомогательных методов jQuery, предназначенных для работы с классами и свойствами CSS. Кроме того, были рассмотрены способы извлечения и изменения HTML-кода и текстового содержимого элементов, а также предусмотренная в jQuery поддержка связывания произвольных данных с элементами путем использования как предназначенного для этого собственного внутреннего механизма, так и атрибутов данных HTML5.

ГЛАВА 9

Работа с событиями

Эта глава посвящена поддержке событий в jQuery. Если события — новая для вас тема, обратитесь к главе 2, в которой были кратко рассмотрены модель событий, используемая в jQuery, и распространение событий по иерархии DOM. Библиотека jQuery предоставляет великолепные возможности для работы с событиями, из которых я бы особо отметил возможность автоматического связывания обработчиков событий с элементами по мере их добавлении в DOM. Перечень тем, рассматриваемых в данной главе, приведен в табл. 8.1.

Таблица 9.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Регистрация функции для обработки одного или нескольких событий	Используйте метод `bind()` или один из прямых методов	1–4, 18, 19, 22
Отмена действия по умолчанию для данного события	Используйте метод `Event.preventDefault()` или метод `bind()` без указания обработчика события	5, 6
Удаление обработчика события из элемента	Используйте метод `unbind()`	7–9
Создание обработчика события, который может выполняться не более одного раза для каждого элемента, с которым он связан	Используйте метод `one()`	10
Автоматическое добавление обработчика события в элементы по мере их вставки в документ	Используйте метод `live()`	11, 12
Удаление обработчика события, который был создан с использованием метода `live()`	Используйте метод `die()`	13
Применение автоматически добавленного обработчика события к заданному элементу DOM и отмена его применения	Используйте методы `delegate()` и `undelegate()()`	14
Вызов обработчиков событий для элемента	Используйте методы `trigger()` или `triggerHandler()` или один из прямых методов	15–17, 20, 21

Обработка событий

В библиотеке jQuery имеется ряд методов, предназначенных для регистрации функций, которые должны вызываться при срабатывании событий на интересующих вас элементах. Эти методы перечислены в табл. 9.1.

Различные разновидности метода `bind()` позволяют указывать функцию, которая должна вызываться при наступлении некоторого события, а поскольку мы имеем

Таблица 9.2. Методы для обработки событий

Метод	Описание
`bind(`*тип_события*`, `*функция*`)` `bind(`*тип_события*`, `*данные*`, `*функция*`)`	Добавляет обработчик событий в элементы, содержащиеся в объекте `jQuery`, с дополнительной возможностью передачи данных обработчику
`bind(`*тип_события*`, `*логическое_значение*`)`	Создает обработчик событий по умолчанию, который всегда возвращает `false`, тем самым предотвращая выполнение действия, предусмотренного по умолчанию. Аргумент *логическое_значение* позволяет управлять всплытием событий
`bind(`*отображение*`)`	Добавляет набор обработчиков событий, заданных объектом отображения, во все элементы, содержащиеся в объекте `jQuery`
`one(`*тип_события*`, `*функция*`)` `one(`*тип_события*`, `*данные*`, `*функция*`)`	Добавляет обработчик событий в каждый из элементов, содержащихся в объекте `jQuery`, с дополнительной возможностью передачи данных обработчику. Обработчик может быть выполнен не более одного раза для каждого из элементов, после чего он отсоединяется от элемента
`unbind()`	Удаляет все обработчики событий из всех элементов, содержащихся в объекте `jQuery`
`unbind(`*тип_события*`)`	Удаляет ранее зарегистрированный обработчик событий из всех элементов, содержащихся в объекте `jQuery`
`unbind(`*тип_события*`, `*логическое_значение*`)`	Удаляет ранее зарегистрированный обработчик событий, всегда возвращающий значение `false`, из всех элементов, содержащихся в объекте `jQuery`
`unbind(Event)`	Удаляет обработчик событий с использованием объекта `Event`

дело с jQuery, то эта функция будет применена по отношению к каждому из элементов, содержащихся в объекте `jQuery`, для которого был вызван метод `bind()`. Простой пример приведен в листинге 9.1.

Листинг 9.1. Использование метода `bind()` для регистрации обработчиков событий

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script type="text/javascript">
        $(document).ready(function() {

            $('img').bind("mouseenter", handleMouseEnter)
                .bind("mouseout", handleMouseOut);

            function handleMouseEnter(e) {
                $(this).css({
                    "border": "thick solid red",
                    "opacity": "0.5"
                });
            };

            function handleMouseOut(e) {
```

```
                $(this).css({
                    "border": "",
                    "opacity": ""
                });
            }
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
    </form>
</body>
</html>
```

В этом примере выбираются все элементы `img` в документе и с помощью метода `bind()` регистрируются обработчики событий `mouseenter` и `mouseout`. В данном

случае обработчики изменяют значения свойств `border` и `opacity` с помощью метода `css()`. При наведении указателя мыши на любой из элементов `img` соответствующее изображение заключается в рамку и становится частично прозрачным, но восстанавливает первоначальное состояние, когда указатель покидает область изображения.

Внутри функции-обработчика переменная `this` ссылается на набор элементов, к которому присоединен обработчик. Передаваемый обработчику объект `Event` является собственным объектом библиотеки jQuery и отличается от объекта `Event`, определенного в спецификации DOM. Свойства и методы объекта `Event` в jQuery описаны в табл. 9.3.

Таблица 9.3. Свойства и методы объекта `Event` в jQuery

Свойство/метод	*Описание*	*Тип возвращаемого значения*
`currentTarget`	Возвращает текущий элемент, событие которого обрабатывается[1]	`HTMLElement`
`data`	Возвращает дополнительные данные, переданные методу `bind()` при регистрации обработчика. За более подробными объяснениями обратитесь к следующему разделу	`object`
`iaDefaultPrevented()`	Возвращает `true`, если для данного объекта событий ранее вызывался метод `preventDefault()`	`boolean`
`isImmediatePropagationsStopped()`	Возвращает `true`, если для данного объекта событий ранее вызывался метод `stopImmediatePropagation()`	`boolean`
`isPropagationsStopped()`	Возвращает `true`, если для данного объекта событий ранее вызывался метод `stopPropagation()`	`boolean`
`originalEvent`	Возвращает первоначальный DOM-объект `Event`	`event`
`pageX` `pageY`	Координаты указателя мыши относительно левого верхнего угла документа	`number`
`preventDefault()`	Отменяет выполнение действий по умолчанию, связанных с данным событием	`void`
`relatedTarget`	Для событий мыши возвращает другой имеющий отношение к событию объект, если таковой имеется. Какой именно будет этот объект, зависит от конкретного события	`HTMLElement`
`result`	Результат, возвращенный обработчиком данного события при его последнем вызове	`object`

[1] Этот элемент не обязательно является источником события, так как последнее могло быть передано ему дочерним элементом посредством механизма "всплытия" событий вверх по DOM-дереву. Для определения источника события необходимо использовать свойство `target`. — *Примеч. ред.*

Окончание табл. 9.3

Свойство/метод	Описание	Тип возвращаемого значения
`stopImmediatePropagation()`	Отменяет выполнение любых других обработчиков, связанных с данным событием	`void`
`stopPropagation()`	Предотвращает всплытие события вверх по иерархии DOM, но разрешает обработчикам, связанным с текущим обрабатываемым элементом, получить данное событие	`void`
`target`	Возвращает элемент, являющийся источником события	`HTMLElement`
`timestamp`	Возвращает время наступления события	`number`
`type`	Возвращает тип события	`string`
`which`	Возвращает информацию о нажатой кнопке или клавише для событий, связанных с мышью или клавиатурой	`number`

Кроме того, для jQuery-объекта `Event` определено большинство свойств, которые имеются у стандартного DOM-объекта `Event`. Поэтому практически во всех ситуациях объект `Event` в jQuery можно рассматривать как расширение функциональности, определенной стандартом DOM[2].

Регистрация функции для обработки нескольких типов событий

Очень часто обработка нескольких типов событий осуществляется с помощью одной функции. Как правило, это делается для событий, являющихся в определенной степени родственными. Типичным примером подобных событий могут служить события `mouseenter` и `mouseout`. Чтобы воспользоваться указанной возможностью, следует вызвать метод `bind()`, задав в качестве первого аргумента список событий, разделенных пробелами. Соответствующий пример приведен в листинге 9.2.

Листинг 9.2. Регистрация функции для обработки нескольких типов событий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind("mouseenter mouseout", handleMouse);

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid red",
                "opacity": "0.5"
}
```

[2] Объект `Event` в jQuery отличается от стандартных объектов `Event` в JavaScript как наличием свойств, измененных для обеспечения кросс-браузерной совместимости, так и наличием дополнительных свойств и методов. — *Примеч. ред.*

```
            if (event.type == "mouseout") {
                cssData.border = "";
                cssData.opacity = "";
}
            $(this).css(cssData);
        }
    });
</script>
...
```

В этом сценарии с помощью единственного вызова метода `bind()` мы указываем, что для всех элементов `img` в документе события `mouseenter` и `mouseout` должны обрабатываться функцией `handleMouse()`. Конечно, того же результата можно было бы добиться, используя цепочку вызовов метода `bind()`.

```
$('img').bind("mouseenter", handleMouse)
        .bind("mouseout", handleMouse);
```

Кроме того, для регистрации обработчиков событий можно использовать объект отображения. Свойствами этого объекта являются имена событий, а значениями свойств — функции, которые должны вызываться при наступлении событий. Пример использования метода `bind()` совместно с объектом отображения приведен в листинге 9.3.

Листинг 9.3. Использование объекта отображения для регистрации обработчиков событий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind({
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

    });
</script>
...
```

В этом примере мы определяем встроенные обработчики событий как элементы объекта отображения. Метод `bind()` осуществляет необходимую привязку обработчиков событий к элементам, используя информацию о событиях и функциях-обработчиках, которую получает через аргументы.

Передача данных обработчику событий

Методу `bind()` можно передать объект, который jQuery впоследствии сделает доступным функции-обработчику через свойство `Event.data`. Это может пригодиться в тех случаях, когда одна и та же функция должна использоваться для обработки событий, источником которых служат разные наборы элементов. Значение свойства `data` можно использовать для того, чтобы определить, какой тип ответной реакции требуется в том или ином случае. Пример определения и использования значения свойства `data` приведен в листинге 9.4.

Листинг 9.4. Передача данных обработчику событий посредством метода `bind()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img:odd')
            .bind("mouseenter mouseout", "red", handleMouse);
        $('img:even')
            .bind("mouseenter mouseout", "blue", handleMouse);

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid " + e.data,
            }
            if (event.type == "mouseout") {
                cssData.border = "";
            }
            $(this).css(cssData);

        }
    });
</script>
...
```

В этом сценарии необязательный аргумент используется для передачи методу `bind()` информации о том, какого цвета рамка должна отображаться при срабатывании события `mouseenter`. Для нечетных элементов `img` рамка будет красной, а для четных — синей. Внутри функции обработки событий данные, которые считываются из свойства `Event.data`, используются для генерации значения CSS-свойства `border`. Конечный результат представлен на рис. 9.1.

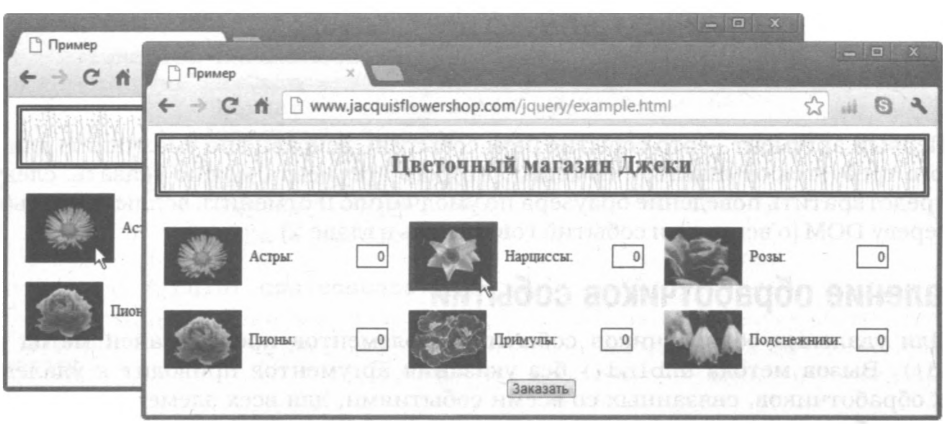


Рис. 9.1. Передача данных обработчику событий через метод `bind()`

Отмена поведения браузера по умолчанию

Как отмечалось в главе 2, для некоторых событий предусмотрены действия по умолчанию, которые выполняются браузером, если источниками событий являются определенные элементы. Хорошим примером этого может служить щелчок на кнопке с атрибутом `type`, имеющим значение `submit`. Если элемент `button` нахо-

дится внутри элемента `form`, то действием по умолчанию для браузера является отправка формы. Чтобы предотвратить выполнение этого действия, следует вызвать метод `preventDefault()` для объекта `Event`, как показано в листинге 9.5.

Листинг 9.5. Отмена действия по умолчанию для события

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('button:submit').bind("click", function(e) {
            e.preventDefault();
        });

    });
</script>
...
```

Пойдем дальше. Обычно вы отменяете действие по умолчанию для того, чтобы вместо него выполнить некоторые другие действия. Например, вы приостанавливаете отправку браузером формы, поскольку хотите сделать это с помощью Ajax (эта тема подробно обсуждается в главах 14 и 15). В приведенном примере можно обойтись без привлечения функции и использовать возможности метода `bind()`, как показано в листинге 9.6.

Листинг 9.6. Использование метода `bind()` для создания обработчика, отменяющего выполнение действия по умолчанию

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('button:submit').bind("click", false);
    });
</script>
...
```

Первый аргумент — это событие (или события), действие по умолчанию для которого требуется отменить; с помощью второго аргумента можно указать, следует ли предотвратить поведение браузера по умолчанию и отменить всплытие события по дереву DOM (о всплытии событий говорилось в главе 2).

Удаление обработчиков событий

Для удаления обработчиков событий из элементов предназначен метод `unbind()`. Вызов метода `unbind()` без указания аргументов приводит к удалению всех обработчиков, связанных со всеми событиями, для всех элементов, содержащихся в объекте `jQuery`, как показано в листинге 9.7.

Листинг 9.7. Удаление всех обработчиков событий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind("mouseenter mouseout", handleMouse);
```

```
        $('img[src*=rose]').unbind();

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid red",
                "opacity": "0.5"
            }
            if (event.type == "mouseout") {
                cssData.border = "";
                cssData.opacity = "";
            }
            $(this).css(cssData);
        }
    });
</script>
...
```

В этом примере сначала со всеми элементами `img` связывается обработчик событий `mouseenter` и `mouseout`, а затем из элемента `img` с атрибутом `src`, содержащим `rose`, с помощью метода `unbind()` удаляются все обработчики событий. Можно прибегнуть к более избирательному подходу и передать события, которые вы хотите открепить от элементов, методу `unbind()` в качестве аргумента, как показано в листинге 9.8.

Листинг 9.8. Избирательное открепление событий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind("mouseenter mouseout", handleMouse);

        $('img[src*=rose]').unbind("mouseout");

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid red",
                "opacity": "0.5"
            }
            if (event.type == "mouseout") {
                cssData.border = "";
                cssData.opacity = "";
            }
            $(this).css(cssData);
        }
    });
</script>
...
```

В этом сценарии открепляется лишь событие `mouseout`, тогда как обработчик события `mouseenter` остается на месте.

Удаление обработчиков событий внутри функций-обработчиков

Наконец, рассмотрим открепление обработчиков событий внутри функции-обработчика. В частности, это может быть полезным в тех случаях, когда событие требуется обработать всего лишь несколько раз. Простой пример того, как это делается, приведен в листинге 9.9.

Листинг 9.9. Открепление события внутри обработчика

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind("mouseenter", handleMouseEnter)
                .bind("mouseout", handleMouseExit)

        var handledCount = 0;

        function handleMouseEnter(e) {
            $(this).css("border", "thick solid red");
        }
        function handleMouseExit(e) {
            $(this).css("border", "");
            handledCount ++;
            if (handledCount == 2) {
                $(this).unbind(e);
            }
        }
    });
</script>
...
```

В функции `handleMouseEvent()` счетчик увеличивается на единицу всякий раз, когда обрабатывается событие `mouseout`. После того как событие обрабатывается во второй раз, объект `Event` передается методу `unbind()` для отмены регистрации функции в качестве обработчика. Всю необходимую для этого информацию jQuery получает из самого объекта.

Установка разового обработчика событий

Метод `one()` позволяет зарегистрировать обработчик событий, который может быть выполнен не более одного раза для одного элемента, после чего удаляется из него. Соответствующий пример представлен в листинге 9.10.

Листинг 9.10. Использование метода `one()` для регистрации разового обработчика событий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').one("mouseenter", handleMouseEnter)
                .one("mouseout", handleMouseOut);

        function handleMouseEnter(e) {
            $(this).css("border", "thick solid red");
        };

        function handleMouseOut(e) {
            $(this).css("border", "");
        };
    });
</script>
...
```

В этом примере обработчики событий mouseenter и mouseout устанавливаются с помощью метода one(). Если пользователь наведет указатель мыши на один из элементов img, а затем сместит его в сторону, то каждый из обработчиков сработает по одному разу, после чего оба обработчика будут удалены из данного элемента и он перестанет реагировать на перемещения мыши. В то же время привязка обработчиков к остальным элементам img сохранится, но это будет длиться лишь до тех пор, пока по отношению к каждому из них не будут совершены аналогичные манипуляции мышью.

Установка обработчиков событий с помощью метода live()

У метода bind() имеется одно ограничение, заключающееся в том, что функции, объявленные в качестве обработчиков событий, не будут связаны с новыми элементами, которые могут впоследствии добавляться в DOM. Соответствующий пример приведен в листинге 9.11.

Листинг 9.11. Добавление элементов после установки обработчиков событий

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img').bind({
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

        $('#row1').append($("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />"));
    });
</script>
...
```

В этом сценарии с помощью метода bind() для всех элементов img устанавливаются обработчики событий mouseenter и mouseout. Далее с помощью метода append() в документ вставляются новые элементы, включая дополнительный элемент img. Во время вызова метода bind() этот элемент еще не существовал, и поэтому обработчики никак с ним не связаны. В результате этого при наведении на них указателя мыши рамками выделяются лишь шесть элементов, тогда как с новым элементом этого не происходит.

В таком простом примере, как этот, проблема легко решается повторным вызовом метода bind(), однако в более сложных случаях следить за тем, какие именно обработчики требуются для различных типов элементов, будет труднее. К счастью, jQuery облегчает жизнь и предлагает набор методов, которые автоматически регистрируют обработчики событий для добавляемых в DOM новых элементов, если они соответствуют селектору. Эти методы перечислены в табл. 9.4.

Таблица 9.4. Методы для автоматической регистрации обработчиков событий

Метод	*Описание*

`live(`*тип_события*`, `*функция*`)`
`live(`*тип_события*`, `*данные*`, `*функция*`)`
`live(`*отображение*`)`
	Добавляет обработчик событий в элементы, соответствующие текущему селектору jQuery, независимо от того, существуют они в данный момент или создаются впоследствии

`die()`	Удаляет все обработчики событий, созданные с помощью метода `live()`
`die(`*тип_события*`)`	Удаляет обработчики событий, созданные методом `live()` для событий указанного типа

`delegate(`*селектор*`, `*тип_события*`, `*функция*`)`
`delegate(`*селектор*`, `*тип_события*`, `*данные*`, `*функция*`)`
`delegate(`*селектор*`, `*отображение*`)`
	Добавляет обработчик событий в элементы (как существующие, так и те, которые будут созданы впоследствии), которые соответствуют селектору, присоединенному к набору элементов, содержащихся в объекте `jQuery`

`undelegate()` `undelegate(`*селектор*`, `*тип_события*`)`	Удаляет обработчики событий, созданные с помощью метода `live()` для событий указанного типа

В листинге 9.12 предыдущий пример представлен в переработанном виде, в котором вместо метода `bind()` используется метод `live()`. Несмотря на кажущуюся незначительность внесенного изменения, оно оказывает весьма существенный эффект. Теперь указанные обработчики событий будут автоматически привязываться к любому элементу, добавляемому в DOM, который соответствует селектору `img`.

Совет. Фактически метод `live()` не добавляет обработчики событий непосредственно в элементы. В действительности он создает обработчики лишь для объекта `document` и осуществляет поиск событий, вызванных элементами, которые соответствуют селектору. В случае обнаружения такого события вызывается соответствующий обработчик. Однако с практической точки зрения проще считать, что метод `live()` добавляет обработчики событий непосредственно в элементы.

Листинг 9.12. Использование метода `live()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').live({
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

        $('#row1').append($("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />"));
    });
```

```
</script>
...
```

В этом примере вновь созданный и добавляемый в документ элемент `img` соответствует селектору объекта `jQuery`, вызывающего метод `live()`, и поэтому связывается с обработчиками событий `mouseenter` и `mouseout`.

Метод `live()` дополняется методом `die()`, который можно использовать для удаления обработчиков и предотвращения возможности их назначения вновь создаваемым элементам, соответствующим селектору. Пример использования метода `die()` приведен в листинге 9.13.

Листинг 9.13. Использование метода `die()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').live({
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

        $('img').die();
    });
</script>
...
```

Предупреждение. Важно, чтобы с методами `live()` и `die()` использовался один и тот же селектор, иначе метод `die()` не сможет удалить результаты работы метода `live()`.

Управление распространением "живых" событий по дереву узлов DOM

С методом `live()` связана одна проблема, заключающаяся в том, что события должны распространиться вверх по дереву иерархии DOM вплоть до элемента `document`, прежде чем объявленные обработчики событий смогут выполниться. Используя метод `delegate()`, можно действовать более целенаправленно и указать, где именно в документе будет располагаться *слушатель события* (event listener). Соответствующий пример приведен в листинге 9.14.

Листинг 9.14. Использование метода `delegate()`

```
<script type="text/javascript">
    $(document).ready(function() {

        $('#row1').delegate("img", {
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
```

```
            mouseout: function() {
                $(this).css("border", "");
            }
        });

        $('#row1').append($("<div class='dcell'/>")
            .append("<img src='carnation.png'/>")
            .append("<label for='carnation'>Гвоздики:</label>")
            .append("<input name='carnation' value='0'
                 required />"));

        $('#row2').append($("<div class='dcell'/>")
            .append("<img src='lily.png'/>")
            .append("<label for='lily'>Лилии:</label>")
            .append("<input name='lily' value='0' required />"));
    });
</script>
...
```

В этом примере метод `delegate()` используется для добавления слушателя событий в элемент с атрибутом `id`, равным `row1`, а указанный селектор обеспечивает отбор элементов `img`, являющихся потомками этого элемента. Функции-обработчики выполняются тогда, когда для одного из этих элементов `img` вызывается событие `mouseenter` или `mouseout` и оно, распространяясь вверх по дереву DOM, достигает указанного элемента `row1`. Если новый элемент `img` добавляется в элемент со значением `id`, равным `row1`, то на него автоматически распространяется действие ранее вызванного метода `delegate()`, чего не происходит, если элемент `img` добавляется в элемент со значением `id`, равным `row2`.

Основное преимущество метода `delegate()` — скорость, что особенно существенно в случае крупных и сложных документов со множеством обработчиков событий[3]. Управляя местоположением точки перехвата событий в документе, вы уменьшаете длину пути, который события должны пройти, распространяясь по дереву DOM, прежде чем смогут быть вызваны обработчики событий.

Совет. Для удаления обработчиков событий, установленных с помощью метода `delegate()`, следует использовать метод `undelegate()`. Метод `die()` работает лишь в паре с методом `live()`.

Вызов обработчиков событий вручную

Для вызова обработчиков событий вручную предусмотрены методы, перечисленные в табл. 9.5.

Таблица 9.5. Методы для вызова обработчиков событий вручную

Метод	*Описание*
`trigger(`*тип_события*`)`	Запускает обработчики событий указанных типов для всех элементов, содержащихся в объекте `jQuery`

[3] В версии jQuery 1.7 вместо метода `delegate()` введен метод `on()`, однако метод `delegate()` по-прежнему поддерживается для совместимости с прежними версиями. — *Примеч. ред.*

Окончание табл. 9.5

Метод	Описание
`trigger(Event)`	Запускает обработчики указанного события для всех элементов, содержащихся в объекте `jQuery`
`triggerHandler(`*тип_события*`)`	Запускает функцию-обработчик для первого из элементов, содержащихся в объекте `jQuery`, без выполнения действий по умолчанию или всплытия событий

Пример запуска обработчиков событий вручную приведен в листинге 9.15.

Листинг 9.15. Запуск обработчиков событий вручную

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind({mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

        $("<button>Запустить</button>").appendTo("#buttonDiv")
            .bind("click", function (e) {
             $('#row1 img').trigger("mouseenter");
             e.preventDefault();
        });

    });
</script>
...
```

В этом сценарии с помощью метода `bind()` устанавливаются два обработчика событий для элементов `img` в документе. Затем мы вставляем в документ элемент `button`, используя для этого метод `append()`, и регистрируем функцию-обработчик для события `click` с помощью метода `bind()`.

После щелчка на созданной кнопке функция-обработчик выбирает элементы `img`, являющиеся потомками элемента со значением `id`, равным `row1`, и использует метод `trigger()` для вызова их обработчиков события `mouseenter`. Результат этого, представленный на рис. 9.2, выглядит так, как если бы указатель мыши был наведен одновременно на все три элемента `img`.

Использование объекта `Event`

Объект `Event` можно использовать для запуска обработчиков событий, прикрепленных к другим элементам. Этот прием удобен для применения внутри обработчиков, как показано в листинге 9.16.

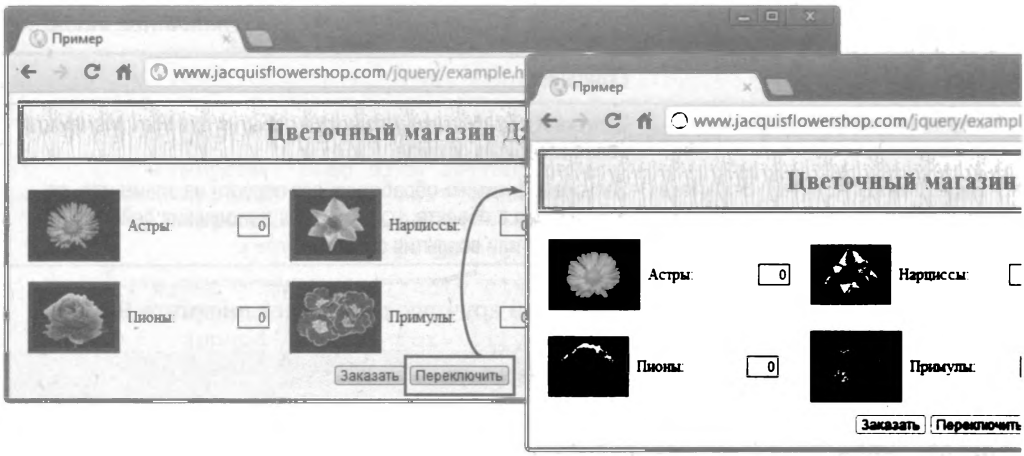


Рис. 9.2. Запуск обработчиков событий вручную

Листинг 9.16. Запуск обработчиков событий вручную с помощью объекта `Event`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#row1 img').bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $('#row2 img').bind("mouseenter", function(e) {
            $(this).css("border", "thick solid blue");
            $('#row1 img').trigger(e);
        });

    });

</script>
...
```

В этом примере мы используем метод `bind()` для добавления красной рамки вокруг элементов `img`, являющихся потомками элемента со значением `id`, равным `row2`, в ответ на наступление события `mouseenter`. Затем то же самое делается в отношении элементов `img`, являющихся потомками элемента со значением `id`, равным `row2`, но с использованием рамки синего цвета. При этом добавляется следующая инструкция:

```
$('#row1 img').trigger(e);
```

Это приводит к тому, что при наведении указателя мыши на один из элементов `img`, являющихся потомками элемента со значением `id`, равным `row2`, срабатывают также обработчики события этого же типа, прикрепленные к элементам `img`, являющимся потомками элемента со значением `id`, равным `row1`. Результат представлен на рис. 9.3.

Такой подход удобно использовать в тех случаях, когда требуется запускать обработчики событий того же типа, что и текущее обрабатываемое событие, однако так же просто можно получить аналогичный результат, указывая тип события.

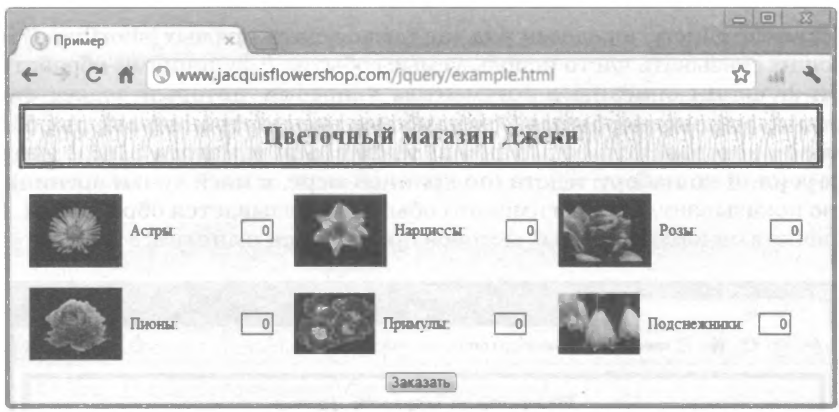


Рис. 9.3. Запуск обработчиков событий с использованием объекта `Event`

Использование метода `triggerHandler()`

Метод `triggerHandler()` также вызывает обработчики событий, однако при этом действия по умолчанию, предусмотренные для данного события, не выполняются и событие не распространяется вверх по дереву DOM. Кроме того, в отличие от метода `trigger()`, метод `triggerHandler()` вызывает функцию-обработчик лишь для первого из элементов, содержащихся в объекте `jQuery`. Пример использования этого метода представлен в листинге 9.17. Другое отличие состоит в том, что результатом метода `triggerHandler()` является результат, возвращаемый функцией-обработчиком. Отсюда следует, что метод `triggerHandler()` не может включаться в цепочки вызовов.

Листинг 9.17. Использование метода `triggerHandler()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#row1 img').bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $('#row2 img').bind("mouseenter", function(e) {
            $(this).css("border", "thick solid blue");
            $('#row1 img').triggerHandler("mouseenter");
        });
    });
</script>
...
```

Результат выполнения этого примера представлен на рис. 9.4.

Использование прямых методов для работы с событиями

В библиотеке jQuery определен ряд так называемых *прямых* (shorthand) методов, позволяющих связывать часто используемые события с функциями-обработчиками. Ниже эти функции описаны с аргументом `function`, который задает функцию-обработчик для данного события. Это наиболее распространенный способ работы с событиями, который полностью эквивалентен вызову метода `bind()`, но требует меньших усилий по набору текста (по крайней мере, с моей точки зрения) и более отчетливо показывает, с каким именно событием связывается обработчик. Пример использования одного из прямых методов представлен в листинге 9.18.

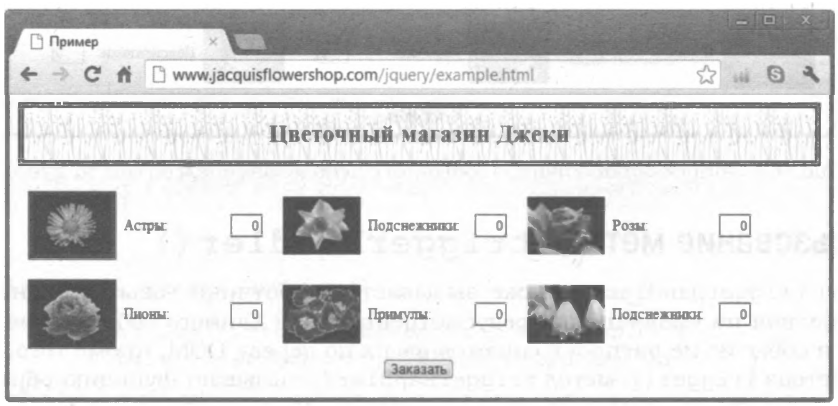


Рис. 9.4. Использование метода `triggerHandler()`

Листинг 9.18. Использование прямого метода для связывания события с функцией-обработчиком

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').mouseenter(function() {
            $(this).css("border", "thick solid red");
        });

    });
</script>
...
```

Это эквивалентно использованию метода `bind()` для привязки события `mouseenter`, как показано в листинге 9.19.

Листинг 9.19. Использование метода `bind()` для события `mouseenter`

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $('img').bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

    });
</script>
...
```

В этом примере нет ничего сложного, и у вас не должно было возникнуть никаких неясностей относительно того, как он работает. Однако прямые методы можно использовать и в качестве аналогов метода `trigger()`. Для этого их следует вызывать без указания аргумента. Соответствующий пример представлен в листинге 9.20.

Листинг 9.20. Использование прямого метода для запуска обработчика событий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $("<button>Запустить</button>")
            .appendTo("#buttonDiv").click(function (e) {
             $('img').mouseenter();
             e.preventDefault();
        });
    });
</script>
...
```

Здесь в документ добавляется кнопка (элемент `button`), после щелчка на которой выбираются элементы `img` и вызываются их обработчики события `mouseenter`. Для полноты картины в листинге 9.21 представлен функционально эквивалентный вариант, в котором используется метод `trigger()`.

Листинг 9.21. Использование метода `trigger()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $("<button>Запустить</button>")
            .appendTo("#buttonDiv").click(function (e) {
            $('img').trigger("mouseenter");
            e.preventDefault();
        });
    });
</script>
...
```

В следующих разделах перечисляются различные категории прямых методов и событий, которым они соответствуют.

Прямые методы для работы с событиями документа

В табл. 9.6 представлены прямые методы, которые используются с объектом `document`.

Метод `ready()` заслуживает отдельного замечания. Ему нет соответствия среди событий DOM, но он чрезвычайно полезен при работе с jQuery. Различные способы использования метода `ready()` продемонстрированы в главе 5, где показано, как задержать выполнение сценария до тех пор, пока не будет полностью построена структура DOM, и как управлять запуском события `ready`.

Таблица 9.6. Прямые методы для работы с событиями документа

Метод	*Описание*
`load(`*функция*`)`	Соответствует событию `load`, которое срабатывает после окончания загрузки всех подчиненных элементов и ресурсов документа
`ready(`*функция*`)`	Срабатывает после обработки всех элементов, содержащихся в документе, и завершения построения DOM-структуры
`unload(`*функция*`)`	Соответствует событию `unload`, которое срабатывает, когда пользователь покидает страницу

Использование прямых методов для работы с событиями браузера

В табл. 9.7 описаны события браузера, источником которых обычно является объект `window` (хотя события `error` и `scroll` могут связываться и с отдельными элементами).

Таблица 9.7. Прямые методы для работы с событиями браузера

Метод	*Описание*
`error(`*функция*`)`	Соответствует событию `error`, которое срабатывает при возникновении проблем с загрузкой внешних ресурсов, например изображений
`resize(`*функция*`)`	Соответствует событию `resize`, которое срабатывает при изменении размера окна браузера
`scroll(`*функция*`)`	Соответствует событию `scroll`, которое срабатывает при использовании полос прокрутки

Использование прямых методов для работы с событиями мыши

В табл. 9.8 приведены прямые методы, предоставляемые jQuery для работы с событиями мыши.

Таблица 9.8. Прямые методы для работы с событиями мыши

Метод	*Описание*
`click(`*функция*`)`	Соответствует событию `click`, которое срабатывает, когда пользователь выполняет щелчок мышью

Окончание табл. 9.8

Метод	Описание
`dbclick(`*функция*`)`	Соответствует событию `dbclick`, которое срабатывает, когда пользователь выполняет двойной щелчок мышью
`focusin(`*функция*`)`	Соответствует событию `focusin`, которое срабатывает при использовании полос прокрутки
`focusout(`*функция*`)`	Соответствует событию `focusout`, которое срабатывает, когда элемент теряет фокус
`hover(`*функция*`)` `hover(`*функция, функция*`)`	Запускается, когда указатель мыши перемещается в область элемента или покидает ее. Если указана только одна функция, она используется в обоих случаях
`mousedown(`*функция*`)`	Соответствует событию `mousedown`, которое срабатывает при щелчке мышью над элементом
`mouseenter(`*функция*`)`	Соответствует событию `mouseenter`, которое срабатывает при наведении указателя мыши на область экрана, занимаемую элементом
`mouseleave(`*функция*`)`	Соответствует событию `mouseleave`, которое срабатывает, когда указатель мыши покидает область экрана, занимаемую элементом
`mousemove(`*функция*`)`	Соответствует событию `mousemove`, которое срабатывает, когда указатель мыши перемещается в пределах области экрана, занимаемой элементом
`mouseout(`*функция*`)`	Соответствует событию `mouseout`, которое срабатывает, когда указатель мыши покидает область экрана, занимаемую элементом
`mouseover(`*функция*`)`	Соответствует событию `mouseover`, которое срабатывает, когда указатель мыши находится в области экрана, занимаемой элементом
`mouseup(`*функция*`)`	Соответствует событию `mouseup`, которое срабатывает при отпускании ранее нажатой кнопки мыши над элементом

Метод `hover()` предлагает удобный способ связывания функции-обработчика с событиями `mouseenter` и `mouseleave`. Если функции передаются два аргумента, то первый из них вызывается в ответ на событие `mouseenter`, а второй — в ответ на событие `mouseleave`. Если вы укажете только одну функцию, то она будет вызываться для обоих событий. Пример использования метода `hover()` представлен в листинге 9.22.

Листинг 9.22. Использование метода `hover()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('img').hover(handleMouseEnter, handleMouseLeave);

        function handleMouseEnter(e) {
            $(this).css("border", "thick solid red");
        };

        function handleMouseLeave(e) {
            $(this).css("border", "");
```

```
        }
    });
</script>
...
```

Использование прямых методов для работы с событиями формы

В табл. 9.9 приведены прямые методы, предоставляемые jQuery для работы с событиями, которые обычно связаны с формами.

Таблица 9.9. Прямые методы для работы для с событиями формы

Метод	*Описание*
`blur(`*функция*`)`	Соответствует событию `blur`, которое срабатывает, когда элемент теряет фокус
`change(`*функция*`)`	Соответствует событию `change`, которое срабатывает при изменении значения элемента
`focus(`*функция*`)`	Соответствует событию `focus`, которое срабатывает, когда элемент получает фокус
`select(`*функция*`)`	Соответствует событию `select`, которое срабатывает при выборе пользователем значения элемента
`submit(`*функция*`)`	Соответствует событию `submit`, которое срабатывает при отправке формы пользователем

Использование прямых методов для работы с событиями клавиатуры

В табл. 9.10 приведены прямые методы, предоставляемые jQuery для работы с событиями клавиатуры.

Таблица 9.10. Прямые методы для работы с событиями клавиатуры

Метод	*Описание*
`keydown(`*функция*`)`	Соответствует событию `keydown`, которое срабатывает, когда пользователь нажимает клавишу на клавиатуре
`keypress(`*функция*`)`	Соответствует событию `keypress`, которое происходит, когда пользователь нажимает и отпускает клавишу на клавиатуре
`keyup(`*функция*`)`	Соответствует событию `keyup`, которое срабатывает, когда пользователь отпускает клавишу на клавиатуре

Резюме

В этой главе были рассмотрены вопросы, связанные с поддержкой событий, которую предоставляет библиотека jQuery. Как и многое другое, что связано с jQuery, основными преимуществами поддерживаемой этой библиотекой функциональности являются ее простота и элегантность. Для создания обработчиков событий и управления ими требуются минимальные усилия. Большое впечатление производит возможность создания "живых" событий, что обеспечивает автоматическое прикрепление обработчиков событий ко вновь создаваемым элементам, соответствующим определенному селектору. Это позволяет значительно сократить время, которое приходится тратить на разрешение проблем, связанных с обработкой событий в веб-приложениях.

ГЛАВА 10

Использование эффектов jQuery

Большая часть функциональности пользовательского интерфейса, связанной с jQuery, содержится в библиотеке jQuery UI, но в ядро библиотеки все же включены некоторые простые эффекты и средства анимации, которые и являются предметом рассмотрения в данной главе. Несмотря на то что я назвал их простыми, с их помощью можно реализовать довольно сложные эффекты. Эти средства предназначены в основном для анимация видимости элементов, однако их можно использовать также для анимации ряда CSS-свойств самыми разными способами. Перечень тем, рассматриваемых в данной главе, приведен в табл. 10.1.

Таблица 10.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Отображение или сокрытие элементов	Используйте метод `show()` или `hide()`	1
Переключение видимости элементов	Используйте метод `toggle()`	2, 3
Анимация видимости элементов	Укажите аргумент `timespan` при вызове метода `show()`, `hide()` или `toggle()`	4
Вызов функции в конце анимации	Укажите функцию обратного вызова в качестве аргумента при вызове метода `show()`, `hide()` или `toggle()`	5–7
Анимация видимости по вертикали	Используйте метод `slidedown()`, `slideUp()` или `slideToggle()`	8
Анимация видимости с использованием непрозрачности	Используйте метод `fadeIn()`, `fadeOut()`, `fadeToggle()` или `fadeTo()`	9–11
Создание пользовательских эффектов	Используйте метод `animate()`	12–14
Остановка и очистка очереди эффектов	Используйте метод `queue()`	15, 16
Просмотр очереди эффектов	Используйте метод `stop()`	17
Вставка задержки в очередь эффектов	Используйте метод `delay()`	18
Вставка пользовательских функций в очередь	Используйте метод `queue()`, указав функцию в качестве аргумента, и убедитесь в выполнении следующей функции в очереди	19, 20
Отключение эффектов анимации	Установите значение свойства `$.fx.off` равным `true`	21

Использование базовых эффектов

Назначением большинства базовых эффектов является простое отображение или сокрытие элементов. Методы, используемые для этой цели, описаны в табл. 10.2.

Таблица 10.2. Методы для создания базовых эффектов

Метод	Описание
`hide()`	Немедленно скрывает все элементы, содержащиеся в объекте `jQuery`
`hide(`*продолжительность*`)` `hide(`*продолжительность*`, `*стиль*`)`	Плавно скрывает элементы, содержащиеся в объекте `jQuery`, в течение заданного времени с возможностью указания стиля анимации
`hide(`*продолжительность*`, `*функция*`)` `hide(`*продолжительность*`, `*стиль*`, `*функция*`)`	Скрывают все элементы, содержащиеся в объекте `jQuery`, в течение заданного времени с возможностью указания стиля анимации и функции, которая вызывается по завершении создания эффекта
`show()`	Немедленно отображает все элементы, содержащиеся в объекте `jQuery`
`show(`*продолжительность*`)` `show(`*продолжительность*`, `*стиль*`)`	Отображают все элементы, содержащиеся в объекте `jQuery`, в течение заданного времени с возможностью указания стиля анимации
`show(`*продолжительность*`, `*функция*`)` `show(`*продолжительность*`, `*стиль*`, `*функция*`)`	Отображают все элементы, содержащиеся в объекте `jQuery`, в течение заданного времени с возможностью указания стиля анимации и функции, которая вызывается по завершении создания эффекта
`toggle()`	Немедленно переключает (отображает, если они скрыты, и скрывает, если они отображаются) видимость элементов, содержащихся в объекте `jQuery`
`toggle(`*продолжительность*`)` `toggle(`*продолжительность*`, `*стиль*`)`	Переключают видимость элементов, содержащихся в объекте `jQuery`, в течение заданного времени с возможностью указания стиля анимации
`toggle(`*продолжительность*`, `*функция*`)` `toggle(`*продолжительность*`, `*стиль*`, `*функция*`)`	Переключают видимость элементов, содержащихся в объекте `jQuery`, в течение заданного времени с возможностью указания стиля анимации и функции, которая вызывается по завершении создания эффекта
`toggle(`*логическое_значение*`)`	Осуществляет одностороннее переключение видимости элементов, содержащихся в объекте `jQuery`

Пример использования методов `show()` и `hide()` без аргументов для создания простейших эффектов приведен в листинге 10.1.

Листинг 10.1. Использование методов `show()` и `hide()` без аргументов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
```

```
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $("<button>Скрыть</button><button>Показать</button>")
                .appendTo("#buttonDiv")
                .click(function(e) {
                    if ($(e.target).text() == "Скрыть") {
                        $('#row1 div.dcell').hide();
                    } else {
                        $('#row1 div.dcell').show();
                    }
                    e.preventDefault();
                 });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
```

```
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
    </form>
</body>
</html>
```

В этом сценарии посредством DOM-манипуляций создаются две кнопки (элементы `button`) и предоставляется функция, которая должна вызываться после щелчка на любой из этих кнопок. Данная функция определяет с помощью метода `text()`, на какой из кнопок был выполнен щелчок, и в зависимости от этого вызывает либо метод `show()`, либо метод `hide()`. В обоих случаях указанная функция вызывается как метод объекта `jQuery`, созданного с помощью селектора `#row1 div.cell`. Таким образом, невидимыми или видимыми будут становиться те объекты `div`, принадлежащие классу `dcell`, которые являются потомками элемента с атрибутом `id`, равным `row1`.

На рис. 10.1 показано, что происходит, когда щелчок выполняется на копке Скрыть.

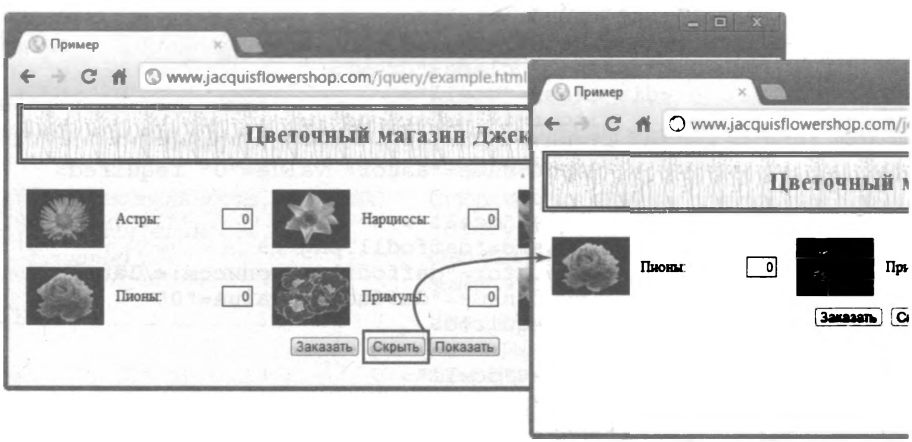


Рис. 10.1. Сокрытие элементов с помощью кнопки Скрыть

Щелчок на кнопке Показать приводит к восстановлению скрытых элементов, как показано на рис. 10.2.

Динамику процесса трудно отразить на рисунках, но есть некоторые моменты, которые заслуживают дополнительных пояснений. Во-первых, в данном случае переход элементов из одного состояния в другое выполняется немедленно, без анимации. Он не сопровождается какими-либо задержками или эффектами, и все элементы появляются или исчезают сразу же после щелчка. Во-вторых, вызов метода `hide()` для элементов, которые уже сделаны невидимыми, равно как и вызов метода `show()` для элементов, которые уже сделаны видимыми, не дает никакого эффекта. Наконец, в-третьих, при сокрытии или отображении элемента одновременно с ним скрываются или отображаются все его потомки.

Совет. Для выбора видимых и невидимых элементов можно использовать селекторы `:visible` и `:hidden`. Более подробные сведения о расширенных CSS-селекторах jQuery содержатся в главе 5.

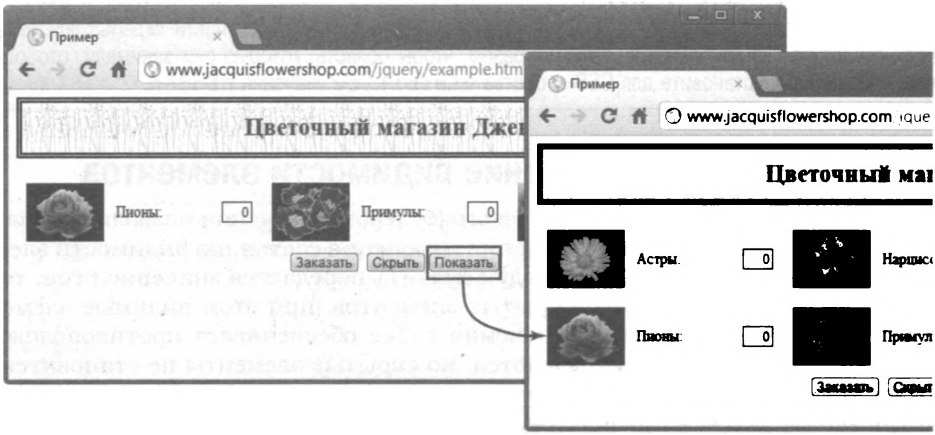


Рис. 10.2. Отображение элементов с помощью кнопки Показать

Переключение видимости элементов

Для перевода элементов из видимого состояния в невидимое и наоборот используется метод `toggle()`. Соответствующий пример приведен в листинге 10.2.

Листинг 10.2. Использование метода `toggle()` для изменения состояния видимости элементов на противоположное

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Переключить</button>").appendTo("#buttonDiv")
            .click(function(e) {
                $('div.dcell:first-child').toggle();
                e.preventDefault();
            });
    });
</script>
...
```

В этом примере мы добавляем в документ единственную кнопку, после щелчка на которой вызывается метод `toggle()`, изменяющий видимость тех элементов `div.dcell`, которые являются первыми дочерними элементами своих родителей. Результат представлен на рис. 10.3.

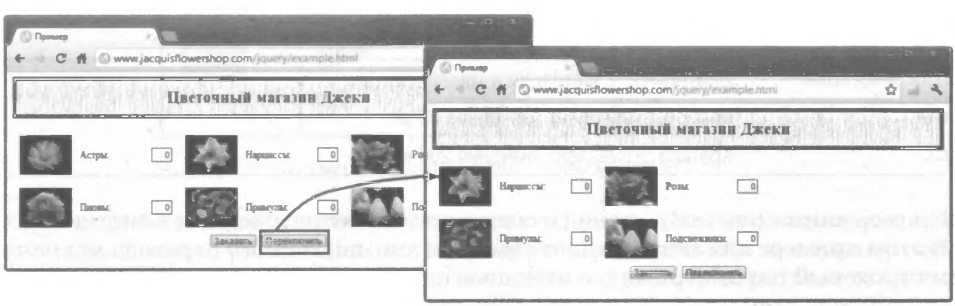


Рис. 10.3. Переключение видимости элементов

Совет. Обратите внимание на сворачивание структуры документа в окрестности скрытых элементов. Если вы хотите скрыть элементы таким образом, чтобы область, которую они занимали, отображалась на странице, установите для CSS-свойства `visibility` значение `hidden`.

Одностороннее переключение видимости элементов

Передача методу `toggle()` логического (булевого) значения позволяет налагать ограничения на возможные направления изменения состояния видимости элементов. Если в качестве аргумента методу `toggle()` передается значение `true`, то это приведет лишь к отображению скрытых элементов (при этом видимые элементы скрываться не будут). Передача значения `false` обеспечивает противоположный эффект: видимые элементы скрываются, но скрытые элементы не становятся видимыми. Пример использования этого варианта вызова метода `toggle()` приведен в листинге 10.3. Должен признаться, я не вижу в этом средстве большой пользы, и включил его в рассмотрение исключительно ради полноты изложения.

Листинг 10.3. Пример использования разновидности метода `toggle()`, предназначенной для одностороннего переключения видимости элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Переключить</button>").appendTo("#buttonDiv")
            .click(function(e) {
                $('div.dcell:first-child').toggle(false);
                e.preventDefault();
            });
    });
    </script>
```

Анимация видимости элементов

Процесс отображения и сокрытия элементов можно анимировать, передавая методу `show()`, `hide()` или `toggle()` параметр, задающий длительность анимации. В этом случае процесс сокрытия или отображения элементов будет осуществляться плавно на протяжении указанного периода. Возможные варианты задания продолжительности периода анимации перечислены в табл. 10.3.

Таблица 10.3. Варианты задания продолжительности периода анимации

Метод	*Описание*
`миллисекунды`	Длительность анимации в миллисекундах
`slow`	Краткий эквивалент значения 600 мс
`past`	Краткий эквивалент значения 200 мс

Пример анимации отображения и сокрытия элементов приведен в листинге 10.4.

В этом примере для задания длительности анимационного перехода мы используем строковый параметр `fast`, с помощью которого указываем, что на переключение видимости элементов `img` в документе отводится 600 мс.

Листинг 10.4. Анимация видимости элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Переключить</button>").appendTo("#buttonDiv")
            .click(function(e) {
                $('img').toggle("fast", "linear");
                e.preventDefault();
            });

    });
</script>
...
```

Совет. Указывая длительность периода анимации в миллисекундах, не заключайте значение в кавычки. Например, следует использовать запись `$('img').toggle(500)`, а не `$('img').toggle("500")`. При наличии кавычек указанное значение будет проигнорировано, и вместо него будет использовано внутреннее значение, установленное по умолчанию.

Вызывая в этом сценарии метод `toggle()`, мы также передаем ему дополнительный аргумент, называемый *функцией смягчения* (easing function) или *стилем смягчения* (easing style), с помощью которого задаем требуемый стиль анимации. Под смягчением здесь понимается смягчение анимационного перехода путем регулирования его скорости в процессе анимации. Доступны два предустановленных стиля смягчения: `swing` и `linear`. Стилю `swing` соответствует медленное начало анимации с последующим ее ускорением и повторным замедлением в конце анимации. Стилю `linear` соответствует постоянная скорость анимации на протяжении всего процесса. Если этот аргумент опущен, то по умолчанию используется значение `swing`.

Результат анимации сокрытия элементов показан на рис. 10.4. Представить анимацию на рисунке довольно трудно, но тем не менее он все же позволяет вам получить некоторое представление о том, что происходит на самом деле.

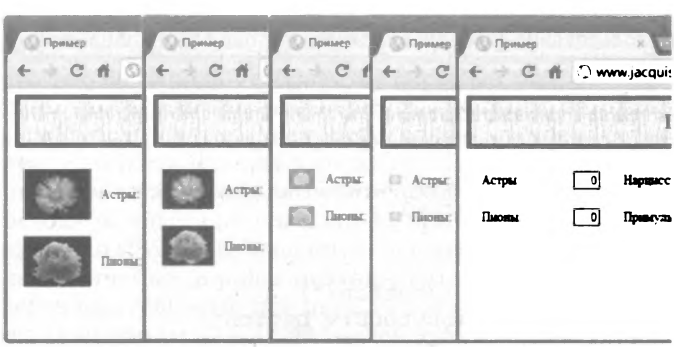


Рис. 10.4. Анимация исчезновения элементов

Как видите, в процессе анимации происходит уменьшение ширины и высоты элементов, а также уменьшение их непрозрачности. Результат повторного щелчка на кнопке Переключить с целью восстановления видимости элементов представлен на рис. 10.5.

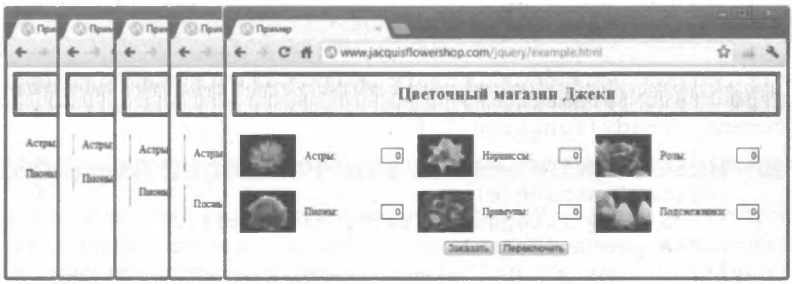


Рис. 10.5. Анимация отображения элементов

При обратном переходе увеличивается лишь размер элементов по вертикали, тогда как их горизонтальный размер скачкообразно восстанавливается перед самым завершением анимационного процесса. Этот факт объясняется неудачным сочетанием способа анимации видимости, используемого jQuery, и компоновки элементов страницы документа в виде таблицы с помощью стилей CSS. Данная проблема не является критической. Ее можно обойти, используя один из других стилей анимации, предназначенных для осуществления анимационных процессов в одном направлении, но, упомянув об этом, я только хотел подчеркнуть, что любая анимация требует тщательного тестирования.

Использование функций обратного вызова в эффектах

Методам `show()`, `hide()` и `toggle()` можно передавать в качестве аргумента функцию, которая будет вызвана по окончании анимации. Эту возможность можно использовать для обновления состояния других элементов, чтобы отразить изменения в их состоянии, как показано в листинге 10.5.

Совет. Если возникает необходимость в последовательном применении нескольких эффектов к одному и тому же элементу, можете воспользоваться обычным приемом jQuery, предполагающим формирование цепочек вызовов. Далее этот вопрос обсуждается более подробно.

Листинг 10.5. Использование функции обратного вызова в виде обработчика события

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var hiddenRow = "#row2";
        var visibleRow = "#row1";

        $(hiddenRow).hide();

        $("<button>Поменять</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                hideVisibleElement();
                e.preventDefault();
            });

        function hideVisibleElement() {
            $(visibleRow).hide("fast", showHiddenElement);
        }
```

```
        function showHiddenElement() {
            $(hiddenRow).show("fast", switchRowVariables);
        }

        function switchRowVariables() {
            var temp = hiddenRow;
            hiddenRow = visibleRow;
            visibleRow = temp;
        }
    });
</script>
...
```

Чтобы сделать этот пример более понятным, функциональность эффекта была распределена между несколькими отдельными функциями. Сначала мы скрываем один из элементов `div`, который служит рядом таблицы элементов в макете страницы, созданном с помощью стилей CSS, и определяем две переменные, с помощью которых отслеживаем, какой именно из рядов является видимым, а какой — невидимым. Далее в документ добавляется кнопка (элемент `button`), после щелчка на которой вызывается функция `hideVisibleElement()`, использующая метод `hide()` для анимации сокрытия видимого ряда элементов.

```
$(visibleRow).hide("fast", showHiddenElement);
```

При этом мы указываем имя функции, которая должна быть вызвана по завершении эффекта, в данном случае — `showHiddenElement`.

Совет. Функции обратного вызова аргументы не передаются, однако переменная `this` будет ссылаться на текущий анимируемый DOM-элемент. Если анимируется несколько элементов, то функция обратного вызова вызывается по одному разу для каждого из них.

В этой функции для анимации восстановления видимости элементов используется метод `show()`:

```
$(hiddenRow).show("fast", switchRowVariables);
```

И вновь мы указываем здесь функцию, которая должна быть вызвана по завершении эффекта. В данном случае ею является функция `switchRowVariables()`, отслеживающая видимость элементов, что позволяет применить анимацию к нужным элементам при выполнении следующего щелчка на кнопке. Теперь при щелчке на кнопке вместо текущего ряда отобразится скрытый ряд, причем анимация выполняется быстро, чтобы длительный переход страницы из одного состояния в другое не раздражал пользователя. Результат работы данного сценария проиллюстрирован на рис. 10.6 (хотя и в этом случае оценить этот эффект по-настоящему вы сможете лишь в том случае, если загрузите пример в браузер).

Процесс перехода осуществляется плавно, причем никаких помех со стороны табличной компоновки CSS не наблюдается, поскольку воздействию подвергаются не индивидуальные ячейки, а ряды таблицы в целом. Обычно необходимости в разбиении эффекта на отдельные функции, как это было сделано в данном примере, не возникает. В листинге 10.6 представлен этот же пример, преобразованный к более компактному виду за счет использования набора встроенных функций.

Листинг 10.6. Использование встроенных функций обратного вызова

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        var hiddenRow = "#row2";
        var visibleRow = "#row1";

        $(hiddenRow).hide();

        $("<button>Поменять</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                $(visibleRow).hide("fast", function() {
                    $(hiddenRow).show("fast", function() {
                        var temp = hiddenRow;
                        hiddenRow = visibleRow;
                        visibleRow = temp;
                    });
                });

                e.preventDefault();
            });
    });
</script>
...
```

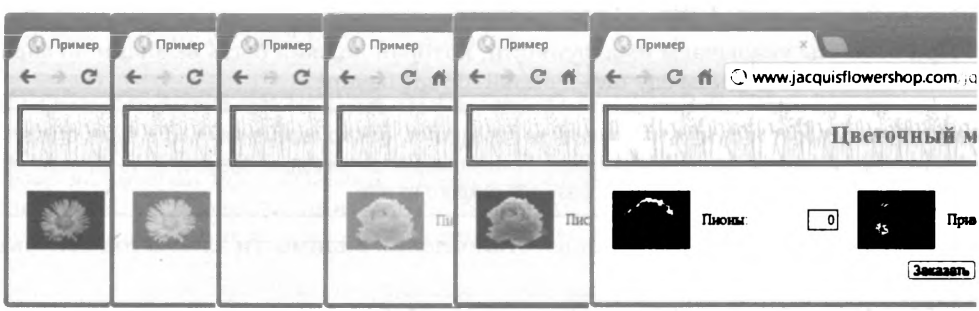


Рис. 10.6. Использование функции обратного вызова для создания цепочки эффектов

Создание циклических эффектов

Используя функции обратного вызова, можно создавать эффекты, выполняющиеся в цикле. Соответствующий пример приведен в листинге 10.7.

Листинг 10.7. Использование функций обратного вызова для создания циклического эффекта

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Пуск</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                performEffect();
                e.preventDefault();
            });

        function performEffect() {
            $('h1').toggle("slow", performEffect)
        }
```

```
    });
</script>
...
```

В этом примере щелчок на кнопке приводит к выполнению функции `performEffect()`. Эта функция использует метод `toggle()` для изменения видимости элемента `h1` в документе и передает ему себя в качестве аргумента. В результате этого элемент `h1` периодически исчезает и появляется.

Совет. Использование текущей функции в качестве функции обратного вызова требует определенной осторожности. В конечном счете вы обязательно исчерпаете стек вызовов JavaScript, и сценарий перестанет работать. Проще всего эта проблема решается с помощью функции `setTimeout()`, которая позволяет организовать обратный вызов целевой функции через определенные промежутки времени без применения вложенных вызовов функций, например так: `$('h1').toggle("slow", setTimeout(performEffect, 1))`. В действительности исчерпать стек вызовов JavaScript довольно трудно, и обычно анимация на странице может выполняться в течение длительного времени, но об этом факторе никогда нельзя забывать.

Используйте эффекты осмотрительно

Я считаю, что циклы, подобные тому, который рассматривался выше, следует использовать весьма умеренно, да и то лишь в тех случаях, когда они служат вполне определенной цели (под этим подразумевается, что они направлены на максимально полное удовлетворение запросов пользователя, а не на демонстрацию вашего умения создавать превосходные эффекты jQuery). В общем случае следует тщательно анализировать, что дает применение того или иного эффекта. Эффекты могут казаться замечательными в процессе разработки, однако их неумеренное использование может свести на нет все усилия, направленные на то, чтобы приложение понравилось пользователям, особенно если оно предназначено для ежедневного использования.

Для примера предположим, что я увлекаюсь бегом (не являясь при этом отличным бегуном). У меня есть специальные наручные часы для бегуна, которые накапливают данные о частоте моего пульса, скорости бега, дистанции, затраченных калориях и еще сотню других показателей. По окончании забега я выгружаю данные на веб-сайт изготовителя часов для сохранения и анализа.

Вот здесь-то и начинается настоящая головная боль. Каждый раз, когда я щелкаю на соответствующей кнопке, содержимое веб-страницы, которое я хочу просмотреть, отображается с использованием длительного эффекта. Я точно знаю, что браузер получил нужные мне данные, поскольку вижу, как они появляются на странице, но должно пройти не менее двух секунд, прежде чем я смогу их прочитать. Возможно, задержка в пару секунд поначалу покажется не очень большой, но она значительна, особенно если каждый раз мне приходится просматривать от пяти до десяти различных характеристик.

Я уверен, что разработчик, проектировавший приложение, думал, что эффекты будут весьма кстати и что они сделают приложение более привлекательным. Однако в процессе работы эффекты оказывают прямо противоположное действие. В данном приложении предусмотрен великолепный набор средств для анализа данных, однако к этому времени я настолько раздражен, что мне уже ничего не хочется делать, и мои данные остаются необработанными. Возможно, я мог бы стать чемпионом по марафону, если бы не эти злополучные эффекты (а вдобавок к ним также пиво и пицца, которые, как оказалось, я поглощаю невероятно часто).

Если вы думаете, что я преувеличиваю (имеются в виду эффекты, а не пиво и пицца), возьмите любой из приведенных в этой главе примеров и установите в нем задержку длительностью в пару секунд. Вы сможете оценить, как невыносимо долго тянется время в ожидании того момента, когда выполнение эффекта завершится.

Суть моего совета состоит в том, что любые эффекты следует применять с осторожностью. Обычно я использую их лишь в случае внесения в DOM таких изменений, которые без применения эффектов воспринимались бы пользователем отрицательно (в качестве примера можно привести внезапное исчезновение элементов со страницы). Если же без эффектов обойтись невозможно, я стремлюсь к тому, чтобы

они длились недолго, обычно не более 200 мс. Я никогда не использую бесконечные циклы, ибо это самый верный способ довести пользователя до головной боли. Я настоятельно рекомендую всегда находить время для того, чтобы как можно лучше понять, как именно пользователи взаимодействуют с приложением или сайтом, и убрать все то, что ничего не дает в плане упрощения решения основных задач. Конечно, хороши сайты, которые имеют привлекательный внешний вид, но привлекательно выглядящие сайты, с которыми пользователю комфортно работать, просто замечательны.

Эффекты плавного изменения высоты элементов

В библиотеке jQuery имеется ряд методов, предназначенных для создания эффектов плавного раскрытия и свертывания элементов путем изменения их высоты (так называемые эффекты скольжения). Эти методы приведены в табл. 10.4.

Таблица 10.4. Методы для создания эффектов скольжения

Метод	*Описание*

`slideDown()`
`slideDown(`*продолжительность*`, `*функция*`)`
`slideDown(`*продолжительность*`, `*стиль*`, `*функция*`)`
	Отображает элементы путем их плавного раскрытия на странице в направлении вниз

`slideUp()`
`slideUp(`*продолжительность*`, `*функция*`)`
`slideUp(`*продолжительность*`, `*стиль*`, `*функция*`)`
	Скрывает элементы путем их плавного свертывания на странице в направлении вверх

`slideToggle()`
`slideToggle(`*продолжительность*`, `*функция*`)`
`slideToggle(`*продолжительность*`, `*стиль*`, `*функция*`)`
	Изменяет состояние видимости элементов путем их плавного свертывания в направлении вверх и раскрытия в направлении вниз

Эти методы анимируют размеры элементов по вертикали. Они принимают те же аргументы, что и методы базовых эффектов. Вы можете передавать им в качестве аргументов длительность эффекта, стиль смягчения и функцию обратного вызова, самостоятельно выбирая тот вариант вызова, который вам подходит. Соответствующий пример представлен в листинге 10.8.

Листинг 10.8. Использование эффектов скольжения

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Переключить</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                $('h1').slideToggle("fast");
                e.preventDefault();
            });
    });
</script>
...
```

В этом примере метод `slideToggle()` используется для создания эффекта попеременного появления и исчезновения элемента `h1` путем изменения его высоты. Результат применения эффекта представлен на рис. 10.7.

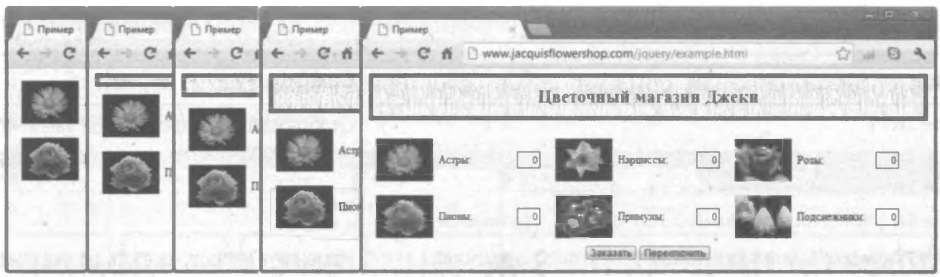


Рис. 10.7. Использование эффекта скольжения для отображения или сокрытия элемента

Рис. 10.8. Создание эффекта путем манипулирования высотой элемента

Рисунок имитирует плавное появление элемента `h1`. В процессе анимации элемент обрезается, а не масштабируется, поскольку, jQuery создает эффект, манипулируя высотой элемента. Чтобы вам стало понятнее, что именно имеется в виду, взгляните на рис. 10.8.

На рисунке крупным планом показаны отдельные стадии процесса появления элемента `h1` на странице. Нетрудно заметить, что размер шрифта текстового содержимого элемента остается неизменным, изменяется лишь размер видимой части текста. Этого, однако, нельзя сказать об изображении элемента, поскольку изображения автоматически масштабируются браузером. Если вы внимательно присмотритесь к рисунку, то заметите, что фоновое изображение во всех случаях показано целиком, однако масштабируется в соответствии с изменением высоты.

Эффекты плавного изменения прозрачности элементов

В этом разделе рассматриваются методы, предназначенные для создания эффектов "растворения" элементов, которые используются для отображения и сокрытия элементов путем изменения их непрозрачности (или, если вам так больше нравится, путем изменения их прозрачности). Эти методы приведены в табл. 10.5.

Таблица 10.5. Методы для создания эффектов растворения

Метод	*Описание*

`fadeOut()`
`fadeOut(`*продолжительность*`)`
`fadeOut(`*продолжительность, функция*`)`
`fadeOut(`*продолжительность, стиль, функция*`)`
	Скрывают элементы путем уменьшения их непрозрачности

Окончание табл. 10.5

Метод	Описание

`fadeIn()`
`fadeIn(`*продолжительность*`)`
`fadeIn(`*продолжительность*`, `*функция*`)`
`fadeIn(`*продолжительность*`, `*стиль*`, `*функция*`)`
	Отображают элементы путем увеличения их непрозрачности

`fadeTo(`*продолжительность*`, `*непрозрачность*`)` `fadeTo(`*продолжительность*`, `*непрозрачность*`, `*стиль*`, `*функция*`)`	Изменяют непрозрачность до указанного уровня

`fadeToggle()`
`fadeToggle(`*продолжительность*`)`
`fadeToggle(`*продолжительность*`, `*функция*`)`
`fadeToggle(`*продолжительность*`, `*стиль*`, `*функция*`)`
	Попеременно отображают и скрывают элементы с использованием непрозрачности

Наборы параметров, используемых в методах `fadeIn()`, `fadeOut()` и `fadeToggle()`, аналогичны наборам параметров, используемых в других методах, создающих эффекты. При вызове этих методов им можно передавать такие параметры, как длительность анимации, стиль анимации и функция обратного вызова, как это было продемонстрировано в предыдущих примерах. Пример использования эффектов прозрачности приведен в листинге 10.9.

Листинг 10.9. Отображение и сокрытие элементов путем изменения их прозрачности

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Переключить</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                $('img').fadeToggle();
                e.preventDefault();
            });
    });
</script>
...
```

Метод `fadeToggle()` применен здесь к элементам `img` частично для демонстрации ограничений этого эффекта. На рис. 10.9 изображено, что происходит на стадии сокрытия элементов.

В отличие от других эффектов, которые изменяют также размер выбранных элементов, эффект прозрачности затрагивает лишь параметр непрозрачности. Это означает, что методы `fade()` обеспечивают плавную анимацию лишь до тех пор, пока элементы не станут полностью прозрачными, после чего jQuery скрывает их, и компоновка страницы изменяется скачкообразно. В некоторых ситуациях такое поведение страницы может раздражать пользователей.

Анимация прозрачности до определенного значения

Для создания анимации, которая прекращается в тот момент, когда параметр непрозрачности достигает определенного значения, используется метод `fadeTo()`. Параметр непрозрачности может принимать значения в интервале от `0` (полная

прозрачность) до 1 (полная непрозрачность). При этом свойство видимости элементов не изменяется, что позволяет избежать скачкообразного изменения компоновки страницы, о котором говорилось выше. Пример использования метода `fadeTo()` приведен в листинге 10.10.

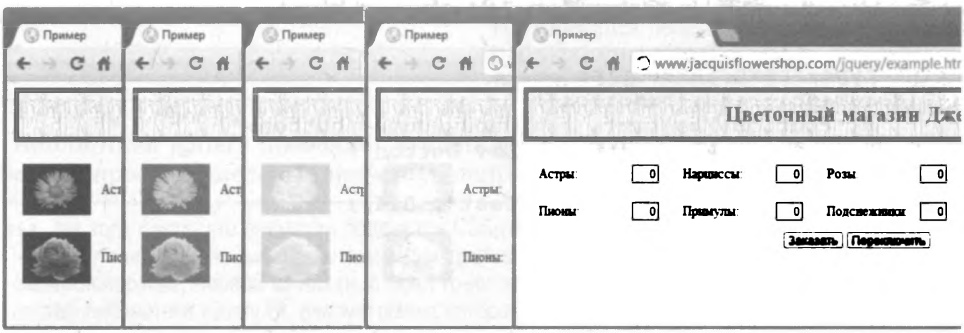


Рис. 10.9. Использование эффекта растворения

Листинг 10.10. Уменьшение непрозрачности до определенного значения

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Уменьшить непрозрачность</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                $('img').fadeTo("fast", 0);
                e.preventDefault();
            });
    });
</script>
...
```

В этом примере мы указали, что непрозрачность элементов должна уменьшаться до тех пор, пока они не станут полностью прозрачными. Производимый эффект в данном случае оказывается тем же, что и при использовании метода `fadeOut()`, однако по завершении анимационного перехода элементы не скрываются. Этот процесс проиллюстрирован на рис. 10.10.

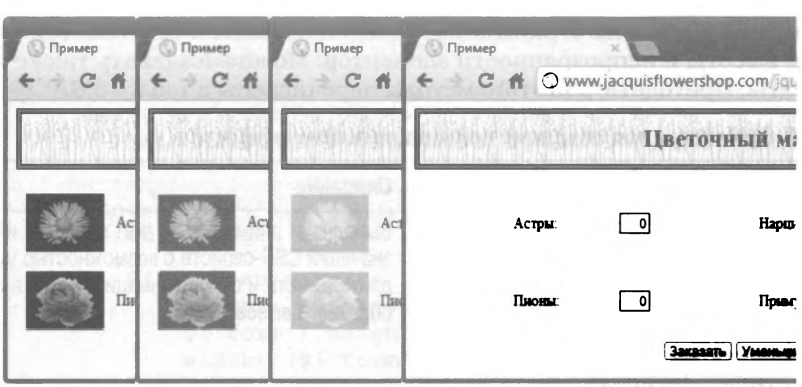


Рис. 10.10. Использование эффекта прозрачности

Уменьшать непрозрачность до нуля вовсе необязательно. Для этого параметра можно указать любое другое значение в пределах допустимого интервала, как показано в листинге 10.11.

Листинг 10.11. Уменьшение непрозрачности до заданного уровня

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Уменьшить непрозрачность</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                $('img').fadeTo("fast", 0.4);
                e.preventDefault();
            });
    });
</script>
...
```

Результат показан на рис. 10.11.

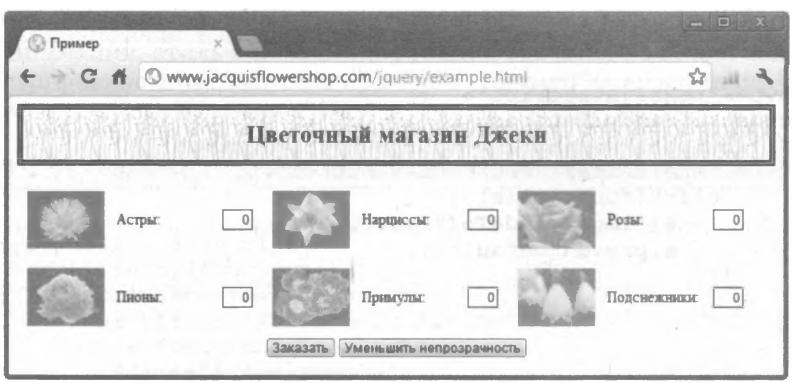


Рис. 10.11. Уменьшение непрозрачности до заданного значения

Создание пользовательских эффектов

Библиотека jQuery не ограничивает вас созданием лишь базовых эффектов анимации высоты и непрозрачности элементов. Можно создавать также собственные эффекты. Пригодные для этого методы перечислены в табл. 10.6.

Таблица 10.6. Методы для создания пользовательских эффектов

Метод	Описание

`animate(`*свойства*`)`
`animate(`*свойства, продолжительность*`)`
`animate(`*свойства, продолжительность, функция*`)`
`animate(`*свойства, продолжительность, стиль, функция*`)`
	Выполняют анимацию за счет плавного изменения значений CSS-свойств с возможностью указания длительности и стиля анимации, а также функции обратного вызова

Окончание табл. 10.6

Метод	Описание
`animate(`*свойства, продолжительность, параметры*`)`	Выполняет анимацию за счет плавного изменения значений CSS-свойств, которые, как и дополнительные параметры, передаются методу в виде объекта с возможностью передачи объекта, содержащего дополнительные параметры, в качестве аргумента

Библиотека jQuery позволяет выполнить анимацию любого свойства, принимающего простые числовые значения, например свойства `height`.

Совет. Из того факта, что анимации поддаются CSS-свойства, выражаемые простыми числовыми значениями, следует невозможность анимации цветов. Для решения этой проблемы существует несколько способов. Первый из них (и, с моей точки зрения, наилучший) — это использование возможностей библиотеки jQuery UI, рассмотрению которой посвящена часть IV. Если же вы не хотите использовать библиотеку jQuery UI, можете воспользоваться встроенной поддержкой анимации свойств CSS, предусмотренной в браузерах. Эти средства обеспечивают неплохую производительность, однако в настоящее время такая поддержка достигается за счет использования различного рода "трюков" и может отсутствовать в браузерах старых версий. Меньше всего мне нравится подход, основанный на использовании подключаемых модулей (плагинов) jQuery. Добиться с их помощью правильной анимации цветов очень трудно, и найти подходящий плагин, который в полной мере устраивал бы меня, мне пока что не удалось. Наиболее надежное из всех известных мне расширений этого типа доступно для загрузки по следующему адресу:

`https://github.com/jquery/jquery-color`

В качестве аргументов метод `animation()` принимает объект отображения, содержащий набор свойств в виде пар "имя–значение", анимацию которых требуется выполнить, и дополнительные параметры. Пример пользовательской анимации приведен в листинге 10.12.

Листинг 10.12. Применение пользовательской анимации

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('form')
            .css({"position": "fixed", "top": "70px",
                "z-index": "2"});
        $('h1')
            .css({"position": "fixed", "z-index": "1",
                "min-width": "0"});

        $("<button>Анимация</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {

                $('h1').animate({
                    height: $('h1').height() +
                    $('form').height() + 10,
                    width: ($('form').width())
                });
```

```
            e.preventDefault();
        });
    });
</script>
...
```

В этом сценарии мы намереваемся анимировать размеры элемента `h1` таким образом, чтобы его фоновое изображение выходило за пределы элемента `form`. Прежде чем это можно будет сделать, необходимо внести некоторые изменения в свойства элементов, затрагиваемые анимацией. Это можно было бы сделать, внеся соответствующие изменения в таблицу стилей, определенную в главе 3. Однако, поскольку книга посвящена jQuery, мы используем для этой цели средства JavaScript. Чтобы упростить управление анимацией, мы позиционируем элементы `form` и `h1` в режиме `fixed` и обеспечиваем отображение элемента `form` поверх элемента `h1` путем установки соответствующих значений свойства `z-index` для этих элементов.

В документ добавлена кнопка, щелчок на которой приводит к вызову элемента `animate()`. Анимируются свойства `height` и `width`, информация о которых получается с помощью методов jQuery. Эффект, производимый анимацией, представлен на рис. 10.12.

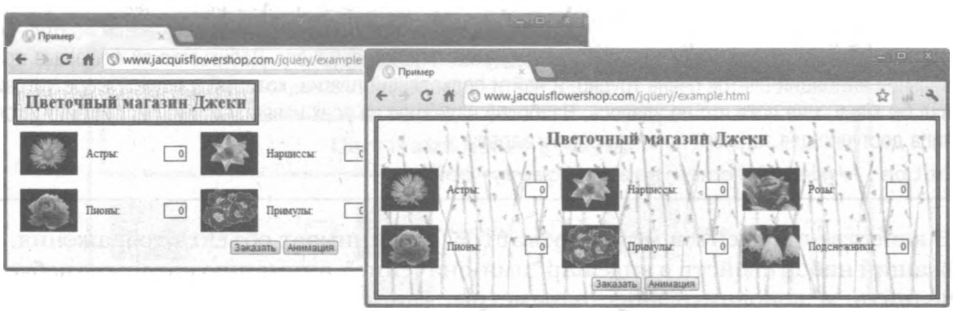


Рис. 10.12. Выполнение пользовательской анимации

На рисунке отображены лишь начальное и конечное состояния анимации, но можете поверить на слово, что, как и в случае других эффектов, переход страницы из одного состояния в другое, выполняемый jQuery, осуществляется плавно, причем способом выполнения этого перехода можно управлять, задавая продолжительность и стиль анимации.

Использование абсолютных целевых значений свойств

Обратите внимание на то, что в процессе создания анимации мы указываем лишь конечные значения свойств. В jQuery при создании пользовательской анимации в качестве отправной точки используются текущие значения анимируемых свойств. В последнем сценарии используются значения свойств, извлеченные с помощью соответствующих методов jQuery, однако для этого существуют и другие возможности. Первая и наиболее очевидная из них — это использование абсолютных значений свойств, как показано в листинге 10.13.

В этом примере мы добавили в анимацию свойство `left`, указав для него абсолютное значение `50` (что будет воспринято jQuery как 50 пикселей). В результате этого элемент `h1` сместится вправо. Конечный результат анимации представлен на рис. 10.13.

Листинг 10.13. Выполнение пользовательской анимации с помощью абсолютных значений свойств

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('form')
            .css({"position": "fixed", "top": "70px",
            "z-index": "2"});
        $('h1')
            .css({"position": "fixed", "z-index": "1",
             "min-width": "0"});

        $("<button>Анимация</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                $('h1').animate({
                    left: 50,
                    height: $('h1').height() + $('form')
                    .height() + 10, width: ($('form').width())
                });

                e.preventDefault();
            });
    });
</script>
...
```

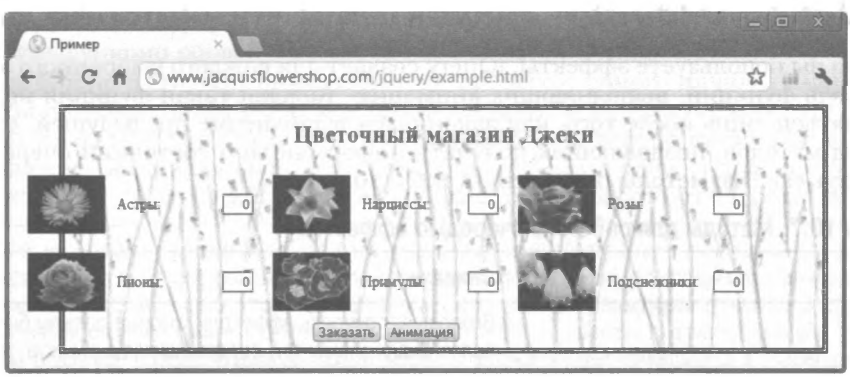


Рис. 10.13. Создание пользовательской анимации с фиксированным конечным значением свойства

Использование относительных целевых значений свойств

При задании целевых значений анимируемых свойств допускается использовать относительные значения. Увеличение значения указывается префиксом `+=`, уменьшение — префиксом `-=`. Пример использования относительных значений приведен в листинге 10.14.

Листинг 10.14. Выполнение пользовательской анимации с помощью относительных значений свойств

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('form')
            .css({"position": "fixed",
                  "top": "70px", "z-index": "2"});
        $('h1')
            .css({"position": "fixed", "z-index": "1",
                  "min-width": "0"});

        $("<button>Анимация</button>")
            .insertAfter("#buttonDiv button")
            .click(function(e) {
                $('h1').animate({
                    height: +=100,
                    width: -=700
                });

                e.preventDefault();
            });
    });
</script>
...
```

Создание очереди эффектов и управление ею

Когда вы используете эффекты, jQuery создает для каждого выбранного элемента очередь функций, выполняющих анимацию. Каждая такая функция начинает выполняться лишь после того, как закончится выполнение предыдущей. Существует ряд методов, позволяющих получать информацию о состоянии очереди или управлять ею. Эти методы приведены в табл. 10.7.

Таблица 10.7. Методы для работы с очередью эффектов

Метод	*Описание*
`queue()`	Возвращает очередь эффектов, которые должны быть выполнены для элементов, содержащихся в объекте `jQuery`
`queue(`*функция*`)`	Добавляет функцию в конец очереди
`dequeue()`	Выполняет и одновременно удаляет из очереди первую из находящихся в ней функций для элементов, содержащихся в объекте `jQuery`
`stop()`	Останавливает текущую анимацию
`stop(`*очистка*`)`	
`stop(`*очистка*`, `*конечный_переход*`)`	
`delay(`*продолжительность*`)`	Вставляет задержку между эффектами, находящимися в очереди

Можете сами создать очередь эффектов, объединив в обычной цепочке вызовов функции, создающие анимационные эффекты, как показано в листинге 10.15.

Листинг 10.15. Создание очереди эффектов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('form').css({"position": "fixed", "top": "70px",
            "z-index": "2"});
        $('h1').css({"position": "fixed", "z-index": "1",
            "min-width": "0"});

        var timespan = "slow";

        cycleEffects();

        function cycleEffects() {
                $('h1')
                .animate({left: "+=100"}, timespan)
                .animate({left: "-=100"}, timespan)
                .animate({height: 223,width: 700}, timespan)
                .animate({height: 30,width: 500}, timespan)
                .slideUp(timespan)
                .slideDown(timespan, cycleEffects);
        }
    });
</script>
...
```

В этом сценарии с помощью обычной цепочки вызовов методов jQuery создается серия анимационных эффектов, применяемых к элементу `h1`. В последнем эффекте в качестве функции обратного вызова используется функция `cycleEffects()`, которая повторно запускает анимационный процесс с самого начала. Циклические эффекты подобного рода очень быстро надоедают, и их следует избегать в повседневной практике. Поначалу они завораживают, но вскоре начинают вызывать раздражение, доставляя пользователям лишь головную боль. В примере этот процесс используется олько для того, чтобы показать, как работают очереди эффектов.

Совет. Тот же результат можно было получить с помощью функций обратного вызова, но в этом случае очередь эффектов не создавалась бы, поскольку функция, запускающая следующую анимацию, начнет выполняться лишь после того, как закончится выполнение предыдущей функции. В случае использования обычной цепочки вызовов методов, как это было сделано в данном примере, анализируются сразу все методы, и анимационные эффекты автоматически помещаются в очередь. Ограниченность этого подхода проявляется в том, что он позволяет работать только с текущим набором выбранных элементов. В то же время, используя функции обратного вызова, можно создавать последовательности эффектов, относящихся к ничем не связанным между собой элементам.

Отображение элементов из очереди эффектов

Для просмотра содержимого очереди эффектов можно использовать метод `queue()`. Правда, пользы от этого будет не очень много, поскольку очередь может содержать один из двух возможных типов объектов данных. Если эффект выполняется, то соответствующим ему элементом очереди является строковое значение `inprogress`. Если же эффект не выполняется, то элементом очереди является функция, которая будет вызываться. Все эффекты jQuery выполняются функцией

doAnimation(), и поэтому вам не удастся определить, какой именно будет следующая анимация. Тем не менее, приступая к работе с очередью, всегда полезно просмотреть ее содержимое. Соответствующий пример приведен в листинге 10.16.

Листинг 10.16. Просмотр содержимого очереди эффектов

```
...
<script type="text/javascript">
    $(document).ready(function() {


        $('h1').css({"position": "fixed", "z-index": "1",
            "min-width": "0"});
        $('form').remove();
        $('<table border=1></table>')
            .appendTo('body').css({
                position: "fixed", "z-index": "2",
                "border-collapse": "collapse", top: 100
            });

        var timespan = "slow";

        cycleEffects();
        printQueue();

        function cycleEffects() {
                $('h1')
                .animate({left: "+=100"}, timespan)
                .animate({left: "-=100"}, timespan)
                .animate({height: 223,width: 700}, timespan)
                .animate({height: 30,width: 500}, timespan)
                .slideUp(timespan)
                .slideDown(timespan, cycleEffects);
        }

        function printQueue() {
            var q = $('h1').queue();
            var qtable = $('table');
            qtable.html("<tr><th>Длина очереди:</th><td>" +
                q.length + "</td></tr>");
            for (var i = 0; i < q.length; i++) {
                var baseString = "<tr><th>" + i + ":</th><td>";
                if (q[i] == "inprogress") {
                    $('table')
                        .append(baseString + "Выполняется</td>
                            </tr>");
                } else if (q[i].name == "") {
                    $('table').append(baseString + q[i] + "</td>
                            </tr>");
                } else {
                    $('table').append(baseString + q[i].name +
                        "</td></tr>");
                }
            }
            setTimeout(printQueue, 500);
```

```
        }
    });
</script>
...
```

В этом примере элемент `form` нам не нужен, в связи с чем мы убираем его из DOM и заменяем простым элементом `table`, который будем использовать для отображения содержимого очереди. В код добавлена также функция `printQueue()`, которая периодически вызывает саму себя через определенные промежутки времени. Эта функции вызывает метод `queue()`, а также отображает в элементе `table` количество элементов в очереди и некоторую информацию о них. Как уже подчеркивалось, особой ценности эта информации не представляет, однако она позволяет получить некоторую общую картину происходящего. Динамику того, как jQuery обрабатывает очередь эффектов, отражает рис. 10.14.

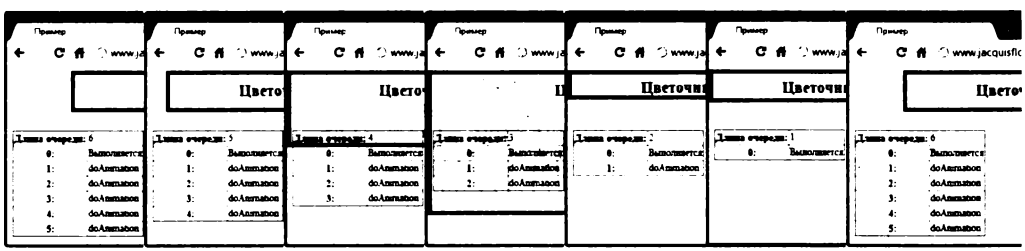


Рис. 10.14. Просмотр содержимого очереди

Этот пример из тех, которые трудно проиллюстрировать с помощью статических рисунков. Поэтому для получения реального представления об этом эффекте лучше всего загрузить пример в браузер. При первом вызове функции `cycleEffects()` в очереди эффектов находится шесть элементов, первый из которых находится в состоянии выполнения. Другие элементы являются экземплярами функции `doAnimation()`. По завершении выполнения каждого эффекта jQuery удаляет соответствующий элемент из очереди. Как только заканчивается выполнение последнего из находящихся в очереди эффектов, происходит повторный вызов функции `cycleEffects()`, восстанавливающей первоначальное состояние очереди, в котором в ней содержится шесть элементов.

Остановка эффектов и очистка очереди

Для остановки выполнения текущей анимации можно использовать метод `stop()`. Можно передать этому методу два дополнительных аргумента, каждый из которых принимает логические значения. Если первый аргумент равен `true`, то все другие эффекты удаляются из очереди и не будут выполняться. Указание значения `true` для второго дополнительного аргумента приведет к тому, что для анимируемых CSS-свойств будут установлены их целевые значения.

По умолчанию оба аргумента имеют значения `false`, означающие, что из очереди удаляется лишь текущий эффект и что анимируемые свойства сохраняют значения, которые они имели на момент остановки выполнения эффекта. Если очередь не очищается, то jQuery переходит к следующему эффекту и выполняет его, как обычно. Пример использования метода `stop()` приведен в листинге 10.17.

Совет. Когда вы вызываете метод `stop()`, функции обратного вызова, связанные с текущим эффектом, не выполняются. Если же метод `stop()` используется для очистки очереди, то не выполняются функции обратного вызова, связанные со всеми эффектами, находящихся в очереди.

Листинг 10.17. Использование метода `stop()`

```
...
<script type="text/javascript">
   $(document).ready(function() {


      $('h1').css({"position": "fixed", "z-index": "1",
          "min-width": "0"});
      $('form').remove();
      $('<table border=1></table>')
          .appendTo('body')
          .css({
              position: "fixed", "z-index": "2",
              "border-collapse": "collapse", top: 100
          });

      $('<button>Стоп</button><button>Пуск</button>')
        .appendTo($('<div/>').appendTo("body")
          .css({position: "fixed", "z-index": "2",
              "border-collapse": "collapse", top: 98, left:220
          })).click(function(e) {
              $(this).text() == "Стоп" ? $('h1')
                  .stop(true, true) : cycleEffects();
          });

      var timespan = "slow";

      cycleEffects();
      printQueue();

      function cycleEffects() {
              $('h1')
              .animate({left: "+=100"}, timespan)
              .animate({left: "-=100"}, timespan)
              .animate({height: 223,width: 700}, timespan)
              .animate({height: 30,width: 500}, timespan)
              .slideUp(timespan)
              .slideDown(timespan, cycleEffects);
      }

      function printQueue() {
          var q = $('h1').queue();
          var qtable = $('table');
          qtable.html("<tr><th>Длина очереди:</th><td>" +
              q.length + "</td></tr>");
          for (var i = 0; i < q.length; i++) {
              var baseString = "<tr><th>" + i + ":</th><td>";
              if (q[i] == "inprogress") {
                  $('table').append(baseString +
                      "Выполняется</td></tr>");
```

```
                } else if (q[i].name == "") {
                    $('table').append(baseString + q[i] +
                        "</td></tr>");
                } else {
                    $('table').append(baseString + q[i].name +
                        "</td></tr>");
                }
            }
            setTimeout(printQueue, 500);
        }
    });
</script>
...
```

Для демонстрации того, как работает метод `stop()`, в документ добавлены две кнопки. После щелчка на кнопке Стоп вызывается метод `stop()`, которому передаются два аргумента `true`. Это приводит к очистке оставшейся части очереди эффектов и немедленной установке целевых значений для анимируемых свойств элемента. Поскольку при использовании метода `stop()` функции обратного вызова не вызываются, циклическое выполнение метода `cycleEffects()` прерывается, и анимация прекращается. После щелчка на кнопке Пуск вызывается метод `cycleEffects()`, что приводит к возобновлению анимации.

Совет. Щелчок на кнопке Пуск во время выполнения анимации не смутит jQuery. Это лишь приведет к тому, что эффекты, используемые методом `cycleEffects()`, добавятся в очередь. Использование функций обратного вызова означает, что длина очереди эффектов будет испытывать незначительные колебания, однако это не отразится на анимации, и она будет продолжаться, как обычно.

Вставка задержки в очередь эффектов

Метод `delay()` позволяет вставить паузу между двумя эффектами, находящимися в очереди. Этот метод принимает один аргумент, который позволяет задать длительность паузы в миллисекундах. Пример использования метода `delay()` представлен в листинге 10.18.

Листинг 10.18. Использование метода `delay()`

```
...
<script type="text/javascript">
    $(document).ready(function() {


        $('h1').css({"position": "fixed", "z-index": "1",
            "min-width": "0"});
        $('form').remove();
        $('<table border=1></table>')
            .appendTo('body')
            .css({
                position: "fixed", "z-index": "2",
                "border-collapse": "collapse", top: 100
            });

        $('<button>Стоп</button><button>Пуск</button>')
          .appendTo($('<div/>').appendTo("body")
            .css({position: "fixed", "z-index": "2",
```

```
            "border-collapse": "collapse", top: 98, left:220
        })).click(function(e) {
            $(this).text() == "Стоп" ? $('h1')
                .stop(true, true) : cycleEffects();
        });

    var timespan = "slow";

    cycleEffects();
    printQueue();

    function cycleEffects() {
            $('h1')
            .animate({left: "+=100"}, timespan)
            .animate({left: "-=100"}, timespan)
            .delay(1000)
            .animate({height: 223,width: 700}, timespan)
            .animate({height: 30,width: 500}, timespan)
            .delay(1000)
            .slideUp(timespan)
            .slideDown(timespan, cycleEffects);
    }

    function printQueue() {
        var q = $('h1').queue();
        var qtable = $('table');
        qtable.html("<tr><th>Длина очереди:</th><td>" +
            q.length + "</td></tr>");
        for (var i = 0; i < q.length; i++) {
            var baseString = "<tr><th>" + i + ":</th><td>";
            if (q[i] == "inprogress") {
                $('table').append(baseString +
                    "Выполняется</td></tr>");
            } else if (q[i].name == "") {
                $('table').append(baseString + q[i] +
                    "</td></tr>");
            } else {
                $('table').append(baseString + q[i].name +
                    "</td></tr>");
            }
        }
        setTimeout(printQueue, 500);
    }
});
</script>
...
```

В этом сценарии в последовательность анимационных эффектов вставлены две паузы, длительность каждой из которых составляет одну секунду. Для обработки задержки метод `delay()` использует анонимную функцию, поэтому при просмотре содержимого очереди в документе вы увидите следующий код.

```
function (next, runner) {
    var timeout = setTimeout(next, time);
    runner.stop = function () {
        clearTimeout(timeout);
    };
}
```

Здесь этот код отформатирован так, чтобы его было удобнее читать, но вам необязательно понимать, как он работает. Мы говорим о нем лишь для того, чтобы вам был понятен источник его происхождения и его появление в очереди не было для вас неожиданностью.

Вставка функций в очередь

Можно добавлять в очередь собственные функции с помощью метода `queue()`, и они будут выполняться точно так же, как и стандартные методы эффектов. Этой возможностью удобно пользоваться для запуска других анимационных процессов, корректного выхода из цепочки анимаций на основе значения внешней переменной и многих других целей. Соответствующий пример содержится в листинге 10.19.

Листинг 10.19. Вставка пользовательской функции в очередь

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('form').css({"position": "fixed", "top": "70px",
            "z-index": "2"});
        $('h1').css({"position": "fixed", "z-index": "1",
            "min-width": "0"});

        var timespan = "slow";

        cycleEffects();

        function cycleEffects() {
                $('h1')
                .animate({left: "+=100"}, timespan)
                .animate({left: "-=100"}, timespan)
                .queue(function() {
                    $('img').fadeTo(timespan, 0).fadeTo(timespan, 1);
                    $(this).dequeue();
                })
                .animate({height: 223,width: 700}, timespan)
                .animate({height: 30,width: 500}, timespan)
                .slideUp(timespan)
                .slideDown(timespan, cycleEffects);
        }
    });
</script>
...
```

Переменная `this` ссылается на объект `jQuery`, для которого был вызван этот метод. Эта возможность весьма полезна, поскольку в некоторый момент вы обязательно должны вызвать в своей функции метод `dequeue()`, чтобы сделать возможным выполнение следующего эффекта или функции в очереди. В этом примере метод `queue()` используется для добавления функции, которая выполняет анимацию прозрачности элемента с использованием метода `fade()`.

Совет. Эффекты, включенные в пользовательскую функцию, добавляются в очереди эффектов для элементов `img`. Каждый элемент имеет собственную очередь, и можно управлять ими независимо одна от другой. Если хотите анимировать одновременно несколько свойств элемента, с очередью которого ведется работа, то следует использовать метод `animate()`. В противном случае эффекты будут лишь последовательно добавляться в очередь.

Можно действовать иначе и передавать функции один аргумент, представляющий следующую функцию в очереди. В подобных случаях для перехода к следующему эффекту нужно вызвать эту функцию самостоятельно, как показано в листинге 10.20.

Листинг 10.20. Передача аргумента в пользовательскую функцию

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('form').css({"position": "fixed", "top": "70px",
            "z-index": "2"});
        $('h1').css({"position": "fixed", "z-index": "1",
            "min-width": "0"});

        var timespan = "slow";

        cycleEffects();

        function cycleEffects() {
                $('h1')
                .animate({left: "+=100"}, timespan)
                .animate({left: "-=100"}, timespan)
                .queue(function(nextFunction) {
                    $('img').fadeTo(timespan, 0)
                        .fadeTo(timespan, 1);
                    nextFunction();
                })
                .animate({height: 223,width: 700}, timespan)
                .animate({height: 30,width: 500}, timespan)
                .slideUp(timespan)
                .slideDown(timespan, cycleEffects);
        }
    });
    </script>
...
```

Совет. Если не вызвать следующую функцию или метод `dequeue`, то дальнейшее выполнение находящихся в очереди эффектов прекратится.

Включение и отключение анимационных эффектов

Анимацию эффектов можно отключить, установив значение свойства `$.fx.off` равным `true`, как показано в листинге 10.21.

Листинг 10.21. Отключение анимации

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $.fx.off = true;

        $('form').css({"position": "fixed", "top": "70px",
            "z-index": "2"});
        $('h1').css({"position": "fixed", "z-index": "1",
            "min-width": "0"});

        var timespan = "slow";

        cycleEffects();

        function cycleEffects() {
                $('h1').animate({left: "+=100"}, timespan)
                .delay(500)
                .animate({left: "-=100"}, timespan)
                .delay(500)
                .queue(function(nextFunction) {
                    $('img').fadeTo(timespan, 0).fadeTo(timespan, 1);
                    nextFunction();
                })
                .delay(500)
                .animate({height: 223,width: 700}, timespan)
                .delay(500)
                .animate({height: 30,width: 500}, timespan)
                .delay(500)
                .slideUp(timespan)
                .delay(500)
                .slideDown(timespan, setTimeout(cycleEffects, 1));
        }
    });
</script>
...
```

При отключенной анимации вызовы методов эффектов приводят к тому, что свойства элементов немедленно принимают целевые значения. Параметры, определяющие длительность анимации, игнорируются, поэтому промежуточная анимация отсутствует. Современные браузеры способны переключаться между состояниями настолько быстро, что заметить такой переход практически невозможно. Именно по этой причине в пример были добавлены вызовы метода `delay()`. Без них указанные скачкообразные переходы были бы просто неразличимы. Необходимо также отметить, что при отключенной анимации наборы эффектов, выполняющихся в цикле, очень быстро приводят к исчерпанию стека вызовов. Во избежание этого в код добавлен вызов метода `setTimeout()`, описанного ранее.

Резюме

В этой главе было продемонстрировано использование эффектов jQuery. Встроенные методы, с помощью которых создаются эффекты, в основном предназначены для плавного изменения видимости элементов различными способами. Однако ваши возможности этим не ограничиваются, поскольку допускается анимация любого из многочисленных CSS-свойств. Наличие доступа к очереди эффектов обеспечивает более точное управление последовательностями эффектов, применяемых к элементам.

ГЛАВА 11

Рефакторинг примера (часть I)

В предыдущих главах каждая из интересующих нас областей функциональности jQuery, таких как обработка событий или DOM-манипуляции, обсуждалась по отдельности. Однако действительные мощь и гибкость jQuery обеспечиваются лишь в случае сочетания всех описанных возможностей. В данной главе совместное использование средств jQuery, с которыми вы к этому времени успели познакомиться, демонстрируется на примере рефакторинга образца документа, лежащего в основе сайта цветочного магазина.

Все изменения, вносимые в документ в этой главе, касаются только элемента `script`. Базовый HTML-код документа остается неизменным. Как это вообще свойственно jQuery, в большинстве случаев для достижения одного и того же результата можно использовать несколько способов. Подходы, предпринятые в данной главе, отражают мои личные предпочтения в использовании различных средств jQuery для манипуляций DOM. Возможно, ваш способ мышления отличается от моего, и вы предпочли бы использовать другие комбинации методов. В действительности это не столь важно, поскольку такого способа использования jQuery, который мог бы считаться единственно правильным, просто не существует.

Пересмотр примера документа

В начале книги был представлен пример простого документа — веб-страницы цветочного магазина. В последующих главах мы обсудили такие вопросы, как выбор элементов в документе, исследование и перестройка лежащей в его основе DOM-модели, обработка событий и применение эффектов к элементам. Прежде чем приступить к улучшению кода примера, вернемся к тому, с чего мы начали, — с базового документа, который приведен в листинге 11.1.

Листинг 11.1. Базовый пример документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            // Сюда будут помещены инструкции jQuery
```

```
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"
                            required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Элемент `script`, которому в книге уделяется основное внимание, выделен в листинге. Пока что этот элемент не содержит ничего, кроме повсеместно используемого в jQuery обработчика события `ready`, наступление которого свидетельствует о готовности документа к работе. Каких-либо других инструкций JavaScript в нем нет. Вид документа без каких-либо "излишеств" в окне браузера представлен на рис. 11.1.

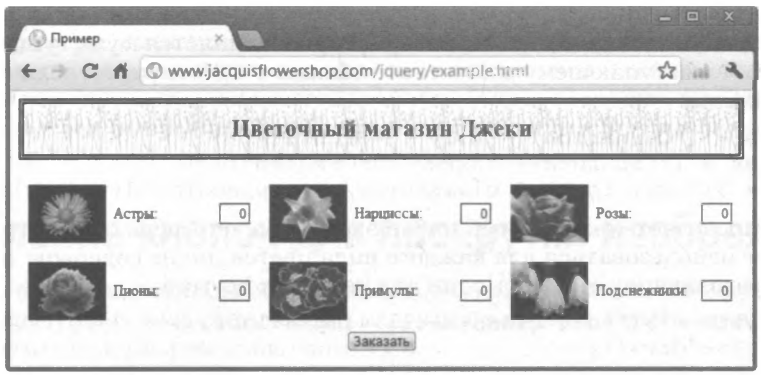


Рис. 11.1. Базовый пример документа

Добавление дополнительных видов цветочной продукции

Первое из изменений, которые мы собираемся внести в документ, касается отображения информации о дополнительных видах цветочной продукции. При этом преследуется двоякая цель. С одной стороны, это позволит продемонстрировать возможности создания элементов в цикле, а с другой — усовершенствовать способ включения в документ других элементов. Элемент `script`, в который добавлен дополнительный код, представлен в листинге 11.2.

Листинг 11.2. Добавление на страницу новых элементов для отображения дополнительных видов цветочной продукции

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNames = ["Carnation", "Lily", "Orchid"];
        var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
        var fRow = $('<div id=row3 class=drow/>')
            .appendTo('div.dtable');
        var fTemplate = $('<div class=dcell><img/><label/>
            <input/></div>');
        for (var i = 0; i < fNames.length; i++) {
            fTemplate.clone().appendTo(fRow).children()
                .filter('img').attr('src', fNames[i] + ".png")
                .end()
                .filter('label').attr('for', fNames[i])
                .text(frNames[i]).end()
                .filter('input').attr({name: fNames[i],
                    value: 0, required: "required"})
        }
    });
</script>
...
```

Здесь введены три дополнительных вида цветов (`Carnation`, `Lily` и `Orchid`)[1] и создан элемент `div` с классом `drow`, который присоединяется к уже существующему элементу `div`, выступающему в качестве таблицы в CSS-модели табличной компоновки страницы.

```
var fNames = ["Carnation", "Lily", "Orchid"];
var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
var fRow = $('<div id=row3 class=drow/>').appendTo('div.dtable');
```

Далее определяется каркасный набор элементов, определяющий структуру, которая будет использоваться для каждого вида цветов, но не содержит никаких атрибутов, позволяющих отличать один вид цветов от другого.

```
var fTemplate = $('<div class=dcell><img/><label/>
    <input/></div>');
```

Каркасные элементы служат простым шаблоном, который копируется для каждого нового вида продукции с использованием названий цветов для добавления соответствующих атрибутов и значений.

```
for (var i = 0; i < fNames.length; i++) {
    fTemplate.clone().appendTo(fRow).children()
        .filter('img').attr('src', fNames[i] + ".png").end()
        .filter('label').attr('for', fNames[i])
            .text(frNames[i]).end()
        .filter('input')
            .attr({name: fNames[i], value: 0,
                required: "required"})
}
```

Для сужения и расширения выбранного набора элементов используются методы `filter()` и `end()`, а для установки значений атрибутов — метод `attr()`. В итоге мы получаем полностью подготовленный набор элементов для представления каждого нового вида цветов, помещаемый в элемент `div` уровня строки таблицы, который, в свою очередь, помещается в элемент уровня таблицы. Конечный результат представлен на рис. 11.2.

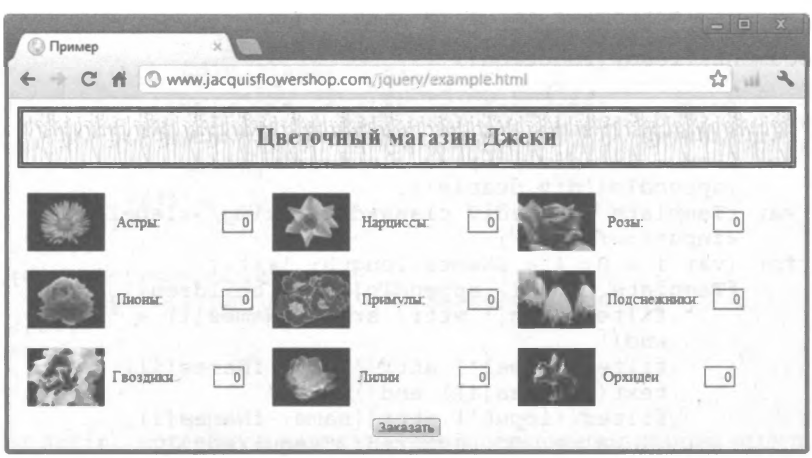


Рис. 11.2. Добавление новых видов цветов на страницу

[1] В сценарий включена дополнительная переменная — `frNames`, — обеспечивающая отображение надписей на русском языке. — *Примеч. ред.*

Одним из преимуществ jQuery, которое со всей очевидностью проявляется в данном примере, является возможность выбора элементов и навигации между ними даже в тех случаях, когда они еще не присоединены к основному документу. На момент клонирования элементов шаблона (`fTemplate`) они еще не являются частью документа, но это никоим образом не препятствует использованию методов `children()` и `filter()` для сужения выбранного набора.

Добавление кнопок для прокрутки изображений

Прежде всего, нам понадобятся левая и правая кнопки, позволяющие переходить к предыдущему или следующему ряду изображений. Реализация этой функциональности представлена в листинге 11.3.

Листинг 11.3. Добавление кнопок для прокрутки изображений

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNames = ["Carnation", "Lily", "Orchid"];
        var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
        var fRow = $('<div id=row3 class=drow/>')
            .appendTo('div.dtable');
        var fTemplate = $('<div class=dcell><img/><label/>
            <input/></div>');
        for (var i = 0; i < fNames.length; i++) {
            fTemplate.clone().appendTo(fRow).children()
                .filter('img').attr('src', fNames[i] + ".png")
                .end()
                .filter('label').attr('for', fNames[i])
                .text(frNames[i]).end()
                .filter('input').attr({name: fNames[i],
                    value: 0, required: "required"})
        }

        $('<a id=left></a><a id=right></a>').prependTo('form')
            .css({
                "background-image": "url(leftarrows.png)",
                "float": "left",
                "margin-top": "15px",
                display: "block", width: 50, height: 50
            }).click(handleArrowPress).hover(handleArrowMouse)

        $('#right')
            .css("background-image", "url(rightarrows.png)")
            .appendTo('form');


        $('#oblock')
            .css({float: "left", display: "inline",
                border: "thin black solid"});
        $('form')
            .css({"margin-left": "auto", "margin-right": "auto",
                width: 885});
```

```
        function handleArrowMouse(e) {
        }

        function handleArrowPress(e) {

        }
    });
</script>
...
```

Мы определяем два новых элемента, присоединяем их к элементу `form` и присваиваем значения ряду свойств с помощью метода `css()`.

```
$('<a id=left></a><a id=right></a>').prependTo('form')
    .css({
        "background-image": "url(leftarrows.png)",
        "float": "left",
        "margin-top": "15px",
        display: "block", width: 50, height: 50
    }).click(handleArrowPress).hover(handleArrowMouse)
```

Ключевым здесь является свойство `background-image`, для которого устанавливается значение `leftarrows.png`. Соответствующее изображение представлено на рис. 11.3.

Это изображение комбинированное и содержит три различных изображения стрелки. Ширина каждой стрелки составляет 50 пикселей, так что, установив для свойств `width` и `height` значение `50`, мы добиваемся того, что каждый раз будет отображаться только одна из стрелок.

Методы `click()` и `hover()` используются для определения функций-обработчиков, обеспечивающих обработку событий `click`, `mouseover` и `mouseexit`.

```
$('<a id=left></a><a id=right></a>').prependTo('form')
    .css({
        "background-image": "url(leftarrows.png)",
        "float": "left",
        "margin-top": "15px",
        display: "block", width: 50, height: 50
    }).click(handleArrowPress).hover(handleArrowMouse)
```

Функции `handleArrowPress()` и `handleArrowMouse()` пока что пустые. Вскоре они будут заполнены инструкциями. На данный момент мы имеем два элемента `arrow`, которые оба отображают стрелки, направленные влево, и расположены рядом один с другим внутри элемента `form`. Элементы `a` были созданы и отформатированы вместе, поскольку в основном они имеют общую конфигурацию, однако теперь самое время передвинуть и немного исправить правую стрелку, что сделано следующим образом.

```
$('#right').css("background-image", "url(rightarrows.png)")
           .appendTo('form');
```

Здесь метод `append()` используется для перемещения элемента в конец элемента `form`, а метод `css()` — для установки такого значения свойства `background-image`, которое обеспечивает использование изображения `rightarrows.png`. Это изображение представлено на рис. 11.4.

Использование комбинированных изображений наподобие этого является распространенным приемом, поскольку это позволяет избежать дополнительных накладных расходов по обработке браузером трех запросов к серверу для получения трех родственных изображений. Вы увидите, как используются такого рода изо-

бражения, когда вскоре мы займемся функцией `handleArrowMouse()`. Результирующий вид страницы в окне браузера на данном этапе представлен на рис. 11.5.

Рис. 11.3. Изображение `leftarrows.png`

Рис. 11.4. Изображение `rightarrows.png`

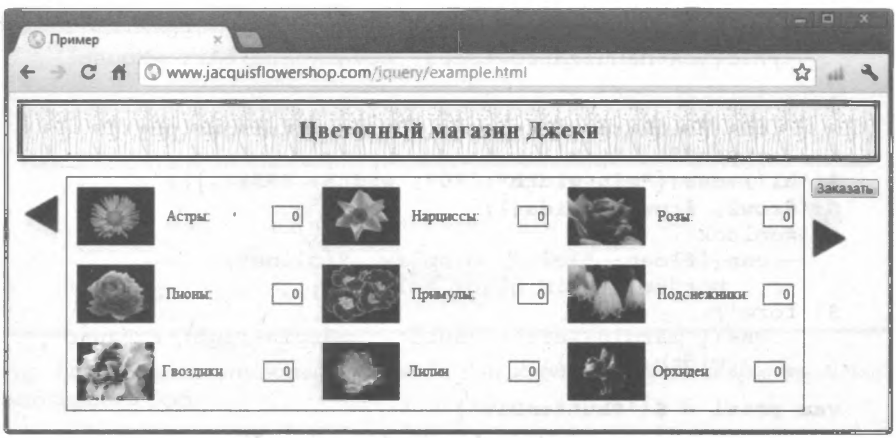


Рис. 11.5. Промежуточное состояние образца документа

Добавление кода для кнопки отправки формы

На рис. 11.5 представлен документ в окне браузера, соответствующий промежуточному состоянию разработки. В документ включены новые возможности, но для того, чтобы их можно было использовать, требуется доработка некоторых существующих элементов. Наиболее важным из них является кнопка Заказать, предназначенная для отправки формы. Соответствующие добавления в сценарий, обеспечивающие работу с этой кнопкой, представлены в листинге 11.4.

Листинг 11.4. Включение в сценарий кода для работы с кнопкой отправки формы

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNames = ["Carnation", "Lily", "Orchid"];
        var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
        var fRow = $('<div id=row3 class=drow/>')
            .appendTo('div.dtable');
        var fTemplate = $('<div class=dcell><img/><label/>
            <input/></div>');
        for (var i = 0; i < fNames.length; i++) {
            fTemplate.clone().appendTo(fRow).children()
                .filter('img').attr('src', fNames[i] + ".png")
```

```
                .end()
                .filter('label').attr('for', fNames[i])
                .text(frNames[i]).end()
                .filter('input').attr({name: fNames[i],
                    value: 0, required: "required"})
        }

        $('<a id=left></a><a id=right></a>').prependTo('form')
            .css({
                "background-image": "url(leftarrows.png)",
                "float": "left",
                "margin-top": "15px",
                display: "block", width: 50, height: 50
            }).click(handleArrowPress).hover(handleArrowMouse)

        $('#right')
            .css("background-image", "url(rightarrows.png)")
            .appendTo('form');
        $('h1').css({"min-width": "0", width: "95%",});
        $('#row2, #row3').hide();
        $('#oblock')
            .css({float: "left", display: "inline",
                border: "thin black solid"});
        $('form')
            .css({"margin-left": "auto", "margin-right": "auto",
                width: 885});

        var total = $('#buttonDiv')
            .prepend("<div>Общий объем заказа: <span id=total>0
                </span></div>")
            .css({clear: "both", padding: "5px"});
        $('<div id=bbox />')
            .appendTo("body").append(total).css("clear: left");

        function handleArrowMouse(e) {
        }

        function handleArrowPress(e) {

        }
    });
</script>
...
```

Чтобы привести документ в соответствие с изменениями в компоновке страницы, вызванными размещением новых кнопок, элемент `div`, содержащий элемент `button` (атрибут `id` которого равен `buttonDiv`), был перемещен в новый элемент `div`, который, в свою очередь, был присоединен к элементу `body`. В результате этого кнопка восстанавливает свое прежнее расположение в нижней части страницы. Также были добавлены элементы `div` и `span`. Эти элементы будут использоваться для отображения общего количества товарных единиц, выбранных пользователем.

```
        var total = $('#buttonDiv')
            .prepend("<div>Общий объем заказа: <span id=total>0
                </span></div>")
            .css({clear: "both", padding: "5px"});
        $('<div id=bbox />')
            .appendTo("body").append(total).css("clear: left");
```

Следующее изменение обеспечивает сокрытие двух рядов с изображениями цветов. Это сделано для того, чтобы впоследствии их можно было отобразить для пользователя при выполнении им щелчков на кнопках прокрутки изображений.

```
$('#row2, #row3').hide();
```

Также был скорректирован стиль элемента h1 для его согласования с измененной компоновкой страницы.

```
$('h1').css({"min-width": "0", width: "95%",});
```

Результат внесения этих изменений отображен на рис. 11.6.

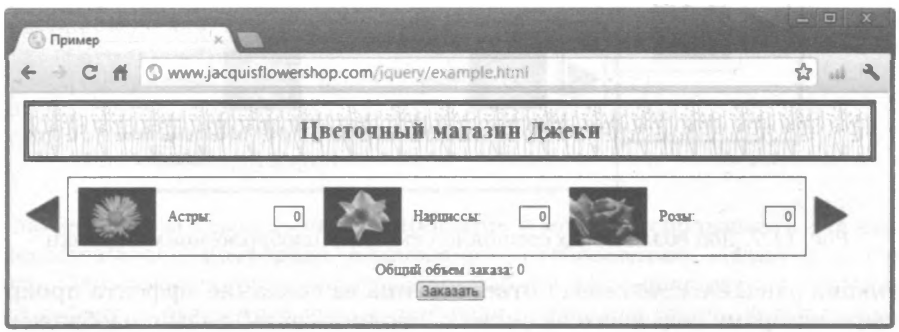


Рис. 11.6. Результат доработки кода для кнопки отправки формы и корректировки CSS

Реализация обработчиков событий для кнопок прокрутки изображений

Наконец-то в документе стали проступать некоторые новые черты, отличающие его от того документа, с которым мы начинали работать. Наш следующий шаг — реализация функций, с помощью которых будет осуществляться обработка событий для кнопок карусельной прокрутки изображений. Мы приступим к этому, начав с событий mouseenter и mouseexit, которые обрабатываются функцией handleArrowMouse(). Код, реализующий эту функцию, представлен в листинге 11.5.

Листинг 11.5. Обработка событий мыши, связанных с кнопками прокрутки изображений

```
...
function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ?
        "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}
...
```

При работе с комбинированными изображениями вся хитрость состоит в использовании свойства background-position для смещения изображения таким образом, чтобы была видна только нужная его часть. Несмотря на то что в каждом из наборов стрелок содержится по три изображения, мы используем только два из них. В обычном состоянии будет отображаться наиболее темная стрелка, тогда как при наведении мыши на элемент вместо нее будет выводиться изображение средней

стрелки. Третья стрелка может быть использована для представления кнопки в состоянии, когда на ней выполнен щелчок или когда она отключена, однако мы не будем усложнять себе жизнь. Два возможных состояния кнопки представлены на рис. 11.7.

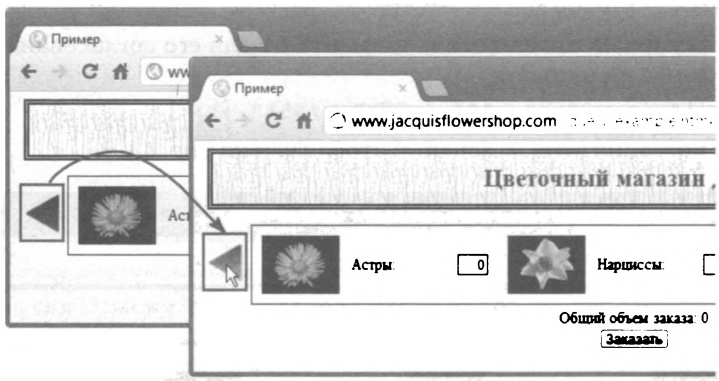


Рис. 11.7. Два возможных состояния кнопки с изображением стрелки

Функция `handleArrowPress()` ответственна за создание эффекта прокрутки, благодаря которому пользователь сможет "пролистывать" ряды с изображениями цветов. Код, реализующий эту функцию, представлен в листинге 11.6.

Листинг 11.6. Реализация функции `handleArrowPress()`

```
...
function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];
    var visibleRow = $('div.drow:visible');
    var visibleRowIndex = jQuery.
        inArray(visibleRow.attr("id"), elemSequence);

    var targetRowIndex;
    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0)
            {targetRowIndex = elemSequence.length -1};
    } else {
        targetRowIndex = (visibleRowIndex + 1) %
            elemSequence.length;
    }

    visibleRow.fadeOut("fast", function() {
        $('#' + elemSequence[targetRowIndex]).fadeIn("fast")});
}
...
```

В этой функции первые три инструкции устанавливают необходимые базовые переменные.

```
var elemSequence = ["row1", "row2", "row3"];
var visibleRow = $('div.drow:visible');
var visibleRowIndex = jQuery.inArray(visibleRow
    .attr("id"), elemSequence);
```

Первая инструкция определяет набор значений атрибута id для элементов row. Во второй инструкции jQuery используется для получения видимого ряда. Эта информация используется далее для определения индекса видимого ряда в массиве значений атрибута id для рядов. (Это делается с помощью вспомогательного метода inArray(), о котором рассказывается в главе 33.) Следовательно, нам известно, какой ряд является видимым и какую позицию он занимает в последовательности рядов. Затем мы вычисляем индекс ряда, который должен отображаться следующим.

```
var targetRowIndex;
if (e.target.id == "left") {
    targetRowIndex = visibleRowIndex - 1;
    if (targetRowIndex < 0)
        {targetRowIndex = elemSequence.length -1};
} else {
    targetRowIndex = (visibleRowIndex + 1) %
        elemSequence.length;
    }
```

В любом другом языке программирования я мог бы использовать для вычисления индекса следующего ряда, подлежащего отображению, оператор деления по модулю. Однако в реализации алгоритма работы этого оператора в JavaScript допущен дефект, в результате чего корректная поддержка отрицательных значений этим оператором не обеспечивается. В связи с этим, если пользователем выполнен щелчок на левой кнопке, мы организуем проверку границ массива вручную, но делаем это с помощью оператора %, если щелчок был выполнен на правой кнопке. Как только текущий отображаемый элемент и элемент, который должен быть отображен вслед за ним, определены, мы используем эффекты jQuery для анимации перехода элементов из одного состояния в другое.

```
visibleRow.fadeOut("fast", function() {
    $('#' + elemSequence[targetRowIndex]).fadeIn("fast")});
```

Для создания эффектов применены методы fadeOut() и fadeIn(), поскольку они отлично сочетаются с компоновкой документа в стиле таблиц CSS. Функция обратного вызова в первом эффекте используется для запуска второго эффекта, причем оба эффекта выполняются с предустановленной длительностью, определяемой строковым параметром fast. Статическое расположение элементов на странице не изменяется, но кнопки в виде стрелок предоставляют пользователю возможность переходить от одного ряда изображений цветов к другому, как показано на рис. 11.8.

Определение общего объема заказа

Последнее из вносимых изменений обеспечивает суммирование заказанных количеств отдельных цветов, введенных в соответствующих полях, и вывод конечного результата непосредственно под рядом, представляющим изображения цветов. Добавленный для реализации этой функциональности код выделен в листинге 11.7 полужирным шрифтом.

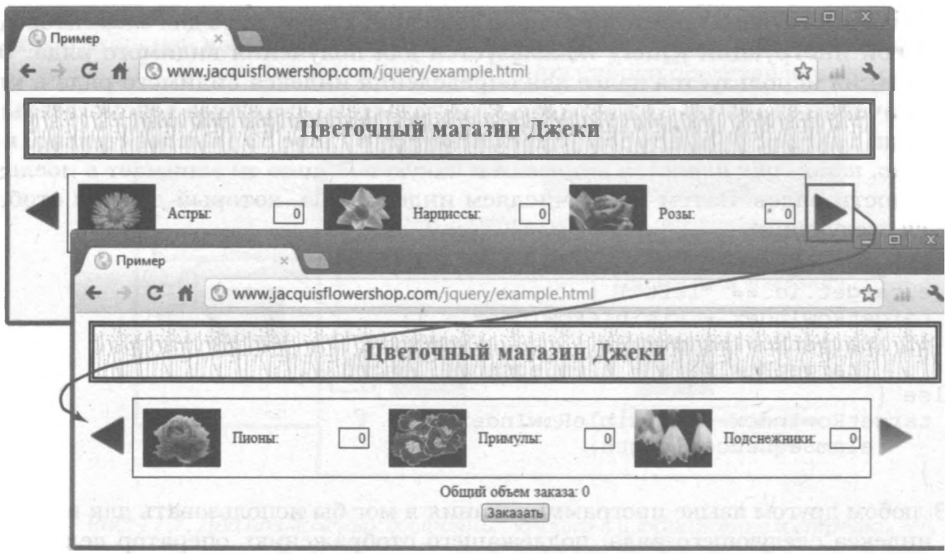


Рис. 11.8. Листание рядов с изображениями различных видов продукции

Листинг 11.7. Суммирование количеств заказанных единиц отдельных видов продукции

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNames = ["Carnation", "Lily", "Orchid"];
        var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
        var fRow = $('<div id=row3 class=drow/>')
            .appendTo('div.dtable');
        var fTemplate = $('<div class=dcell><img/><label/>
            <input/></div>');
        for (var i = 0; i < fNames.length; i++) {
            fTemplate.clone().appendTo(fRow).children()
                .filter('img').attr('src', fNames[i] + ".png")
                .end()
                .filter('label').attr('for', fNames[i])
                .text(frNames[i]).end()
                .filter('input').attr({name: fNames[i],
                    value: 0, required: "required"})
        }

        $('<a id=left></a><a id=right></a>').prependTo('form')
            .css({
                "background-image": "url(leftarrows.png)",
                "float": "left",
                "margin-top": "15px",
                display: "block", width: 50, height: 50
            }).click(handleArrowPress).hover(handleArrowMouse)

        $('#right')
            .css("background-image", "url(rightarrows.png)")
```

```
            .appendTo('form');

        $('h1').css({"min-width": "0", width: "95%",});
        $('#row2, #row3').hide();
        $('#oblock')
            .css({float: "left", display: "inline",
                border: "thin black solid"});
        $('form')
            .css({"margin-left": "auto", "margin-right": "auto",
                width: 885});

        var total = $('#buttonDiv')
            .prepend("<div>Общий объем заказа: <span id=total>0
                </span></div>")
            .css({clear: "both", padding: "5px"});
        $('<div id=bbox />')
            .appendTo("body").append(total).css("clear: left");

        $('input').change(function(e) {
            var total = 0;
            $('input').each(function(index, elem) {
                total += Number($(elem).val());
            });
            $('#total').text(total);
        });

        function handleArrowMouse(e) {
            var propValue = e.type == "mouseenter" ?
                "-50px 0px" : "0px 0px";
            $(this).css("background-position", propValue);
        }

        function handleArrowPress(e) {
            var elemSequence = ["row1", "row2", "row3"];
            var visibleRow = $('div.drow:visible');
            var visibleRowIndex = jQuery.
                inArray(visibleRow.attr("id"), elemSequence);

            var targetRowIndex;
            if (e.target.id == "left") {
                targetRowIndex = visibleRowIndex - 1;
                if (targetRowIndex < 0)
                    {targetRowIndex = elemSequence.length -1};
            } else {
                    targetRowIndex = (visibleRowIndex + 1) %
                        elemSequence.length;
            }

            visibleRow.fadeOut("fast", function() {
                $('#' + elemSequence[targetRowIndex])
                .fadeIn("fast")});
            }
    });
</script>
...
```

С помощью добавленного кода в документе выбираются элементы input и регистрируется обработчик событий, который получает значения из всех текстовых полей, суммирует их и устанавливает полученную сумму в качестве содержимого добавленного ранее элемента span. Конечный результат представлен на рис. 11.9.

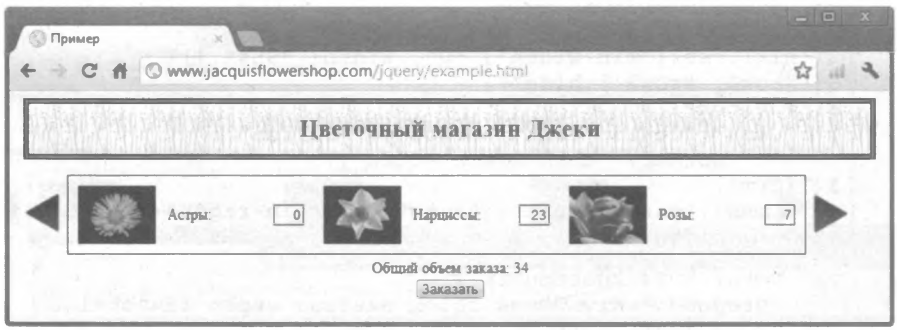


Рис. 11.9. Отображение суммарного количества заказанных единиц товара

Отключение JavaScript

В этой главе мы заметно преобразовали исходный документ, но при этом все изменения были выполнены средствами jQuery. Это означает, что мы создали фактически два документа: один — для браузеров, в которых разрешено выполнение сценариев JavaScript, и второй — для браузеров, в которых это запрещено. На рис. 11.10 показано, в каком виде предстанет перед вами документ в окне браузера при отключенном JavaScript.

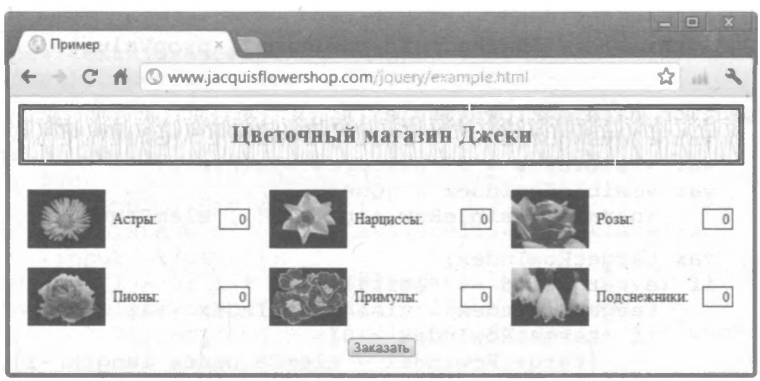


Рис. 11.10. Внешний вид документа в окне браузера в случае запрета выполнения сценариев JavaScript

Мы вернулись к тому, с чего начали. Если вы проявите предусмотрительность и не пожалеете времени на планирование своих действий, то сможете предложить клиентам, в браузерах которых JavaScript отключен, некоторый набор функциональных возможностей, который позволит им взаимодействовать с вашей веб-страницей или приложением. В целом это неплохая идея. Дело в том, что во многих крупных компаниях, где управление всеми информационными ресурсами осуществляется централизованно, выполнение сценариев JavaScript запрещено из соображений безопасности. (Проработав в подобных организациях в течение многих лет, я пришел к выводу, что на самом деле такая политика не способна полностью предотвратить использование JavaScript персоналом. Она лишь заставляет людей искать любые доступные лазейки или пути для обхода такого запрета.)

Резюме

В этой главе методики, рассмотренные в предыдущих главах, были объединены для внесения улучшений в пример документа, с которым мы работаем. Мы добавили новое содержимое программным путем, создали простой механизм листания каталога товаров, а также ввели новый элемент, отображающий общий объем заказа. При этом все изменения вносились за счет изменения DOM-структуры и CSS-свойств элементов, но таким образом, чтобы даже те пользователи, в браузерах которых выполнение сценариев запрещено, по-прежнему имели в своем распоряжении документ, пригодный для работы.

В следующей части мы продолжим наращивать возможности работы с нашим образцом документа, расширяя его функциональность за счет использования других возможностей jQuery. В основном все это будет делаться на базе исходного образца документа, чтобы каждый раз наше внимание было сосредоточено на каком-то одном средстве. Однако в главе 16 мы вновь переработаем документ для улучшения его функциональности.

Часть III

Работа с данными и Ajax

ГЛАВА 12

Использование шаблонов данных

В этой главе вы познакомитесь со своим первым подключаемым модулем (плагином) jQuery — *jQuery Templates*. Он обеспечивает возможность использования шаблонов, упрощающих генерацию HTML-элементов из объектов данных JavaScript.

Чтобы не возникало недопонимания, хочу предупредить, что данный модуль не относится к числу активно разрабатываемых или поддерживаемых в настоящее время, и команда разработчиков jQuery не рекомендует его применять. Это не означает, что вы не должны его использовать, однако я счел своим долгом сказать вам об этом, прежде чем вы будете включать его в свои проекты. Я был бы рад порекомендовать вам какой-нибудь другой активно разрабатываемый вариант, однако найти хотя бы близкую по своим возможностям замену jQuery Templates мне пока что не удалось. Но даже при упомянутом отношении к нему разработчиков этот модуль все еще остается наилучшим.

История модуля jQuery Templates довольно необычна. В свое время Microsoft и команда разработчиков jQuery объявили, что трем подключаемым модулям, разработанным компанией Microsoft, присвоен статус "официальных", чего до того не удостаивался ни один из подключаемых модулей. Спустя какое-то время команда jQuery объявила об отказе от использования этих модулей и лишении их статуса официальных, а также о своих планах относительно замены их другой функциональностью. Предполагаемая замена должна была войти в состав библиотеки jQuery UI (которой посвящена часть IV). Прискорбно, но факт: ничего из того, что было обещано, пока еще не предоставлено, а отвергнутые плагины по-прежнему доступны и широко используются (особенно это относится к подключаемому модулю для работы с шаблонами). Совершенно очевидно, что решение о том, использовать ли код, применение которого не рекомендовано, каждый принимает самостоятельно, но лично мне нравится функциональность, обеспечиваемая шаблонами, и я часто ее использую. При этом я исхожу из того, что могу в любой момент заглянуть в исходный код и устранить любую серьезную проблему, если она возникнет, а то, что иногда все-таки приходится искать обходные пути для преодоления незначительных затруднений, окупается выгодами, которые дает использование шаблонов. Перечень тем, рассматриваемых в данной главе, приведен в табл. 12.1.

Таблица 12.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Генерация элементов с помощью шаблонов	Определите шаблон в элементе `script` и примените его, используя метод `tmpl()`	1–6

Окончание табл. 12.1

Задача	Решение	Листинг
Назначение элементов, сгенерированных на основе шаблона, различным родителям	Либо разбейте исходные данные на несколько наборов и выполните рендеринг шаблона несколько раз, либо используйте методы `slice()`, `filter()` и `end()` для разделения сгенерированных элементов	7–9
Вставка результата вычисления выражения в шаблон	Поместите выражение в дескриптор шаблона `${...}`	10
Доступ к объекту данных в шаблоне	Используйте переменную `$data`	11
Выбор элементов внутри шаблона	Используйте функцию jQuery `$()`	12
Передача параметров шаблону	Передайте объект отображения методу `tmpl()` и используйте переменную `$item` для доступа к параметрам как к свойствам	13
Рендеринг вложенного шаблона	Используйте дескриптор шаблона `{{tmpl}}`	14–16
Создание условных областей в шаблоне	Используйте дескрипторы шаблона `{{if}}` и `{{else}}`	17–19
Управление рендерингом шаблона для элементов массива	Используйте дескриптор шаблона `{{each}}`	20, 21
Отключение экранирования специальных символов при вставке данных в шаблон	Используйте дескриптор шаблона `{{html}}`	22, 23
Повторный рендеринг элементов с использованием других шаблонов или значений данных	Используйте метод `$.tmplItem()` для получения объекта элемента шаблона. Используйте свойства этого объекта для изменения шаблона или данных и вызовите метод `update()` для повторной генерации содержимого	24–26

Для чего нужны шаблоны

Шаблоны данных решают вполне определенную задачу: они обеспечивают программную генерацию элементов на основе свойств и значений объектов JavaScript, тем самым избавляя нас от выполнения рутинных операций. Для этого существуют и другие способы. В действительности нечто весьма похожее мы уже делали в главе 11, когда создавали элементы для представления дополнительных видов цветочной продукции в образце документа. Соответствующая часть сценария из этой главы приведена в листинге 12.1.

Листинг 12.1. Создание элементов программным путем

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNames = ["Carnation", "Lily", "Orchid"];
        var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
        var fRow = $('<div id=row3 class=drow/>')
            .appendTo('div.dtable');
```

```
        var templates = $('<div class=dcell><img/><label/>
            <input/></div>');
        for (var i = 0; i < fNames.length; i++) {
            fTemplate.clone().appendTo(fRow).children()
                .filter('img').attr('src', fNames[i] + ".png")
                .end()
                .filter('label').attr('for', fNames[i])
                .text(frNames[i]).end()
                .filter('input').attr({name: fNames[i],
                    value: 0, required: "required"})
        }
    });
</script>
...
```

Проблема в том, что такой подход плохо масштабируется. Если даже эти инструкции читаются нелегко, то с усложнением элементов трудности только возрастают. По моему мнению, проблемы с удобочитаемостью кода возникают из-за того, что задачи, связанные сугубо с HTML-элементами, пытаются решать средствами JavaScript. К счастью, как будет показано далее, в библиотеке шаблонов jQuery основной упор делается именно на работе с HTML-кодом, и, если только вам не требуется нечто весьма специфическое, она позволяет минимизировать объем кода, необходимого для генерации элементов на основе данных.

Если взглянуть на эту ситуацию несколько шире, то можно прийти к заключению, что интеграция данных в документ является проблемой общего характера. В моих проектах эта проблема проявляется в двух ситуациях. Во-первых, мне приходится сталкиваться с ней при работе с уже существующими системами, которые содержат данные, управляющие моим веб-приложением. Я *мог бы* — и для этого существуют замечательные технологии — получать данные и интегрировать их в документ на сервере, но это означало бы, что моя серверная система будет тратить массу времени на выполнение работы, которую вполне можно было бы переложить на браузеры. Если вам когда-либо приходилось создавать и сопровождать веб-приложения, обрабатывающие большие объемы данных, то вам должно быть известно, что связанные с этим накладные расходы весьма ощутимы, и поэтому к любой возможности минимизировать объем обрабатываемых данных нужно всегда относиться самым серьезным образом.

Второй причиной, заставляющей меня интегрировать данные в документы, является то, что мое веб-приложение получает данные посредством Ajax-запросов в ответ на действия пользователя. О поддержке Ajax в jQuery подробно рассказывается в главах 14 и 15, а пока что вам будет вполне достаточно знать лишь то, что с помощью библиотеки Ajax можно получать и отображать данные с сервера без перезагрузки всей страницы. Эта мощная методика, которая в настоящее время приобрела огромную популярность, прекрасно сочетается с шаблонами данных.

Настройка библиотеки jQuery Templates

Прежде чем использовать шаблоны jQuery, нужно загрузить библиотеку jQuery Templates и подключить ее к своему документу. Эта библиотека доступна для загрузки по следующему адресу:

`https://github.com/jquery/jquery-tmpl`

Распакуйте архив и скопируйте файл `jquery.templ.js` (версия для разработки) или `jquery.templ.min.js` (версия для развертывания) на свой веб-сервер, жела-

тельно в тот же каталог, в котором находится основной JavaScript-файл библиотеки jQuery.

Следующее, что необходимо сделать, — добавить в образец документа элемент `script`, подключающий библиотеку шаблонов, как показано в листинге 12.2.

Листинг 12.2. Добавление библиотеки шаблонов в образец документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            // Сюда будет помещаться код примера

        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2"class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Мы будем использовать листинг 12.2 в качестве образца документа в этой главе. Наверное, вы заметили, что от первоначального варианта он отличается не только тем, что в него добавлена библиотека шаблонов, но и тем, что из него удалены элементы, соответствующие различным видам цветочной продукции. Это сделано специально для того, чтобы мы могли восстанавливать эти элементы в документе различными способами с помощью библиотеки шаблонов. Внешний вид исходного документа в окне браузера на данном этапе представлен на рис. 12.1.

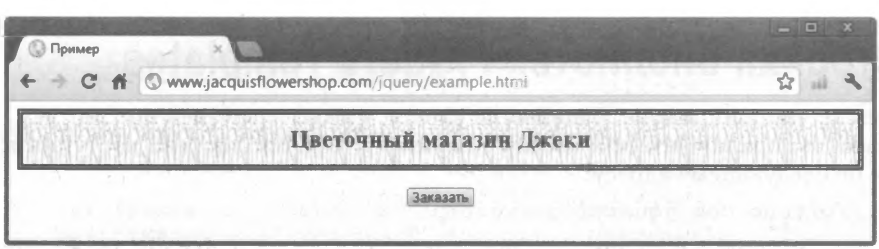


Рис. 12.1. Исходный документ

Предупреждение. Ранее я уже рассказал о том, каким образом можно использовать средства jQuery для улучшения и переработки документа, чтобы даже те пользователи, в браузерах которых выполнение сценариев JavaScript не разрешено, все еще могли продолжать пользоваться приложением. Следует всегда стремиться к реализации именно такого подхода, однако описываемые в данной главе методики с этим плохо согласуются. Идея привлечения JavaScript для создания элементов на основе данных с помощью шаблонов в значительной степени противоречит идее параллельного создания эквивалентных сценариев, не использующих JavaScript. В конце концов, если вы готовите вариант документа, в котором уже содержатся элементы, порожденные данными, то в использовании шаблонов нет необходимости. Я большой приверженец использования альтернативных вариантов документа в расчете на тех пользователей, которые лишены возможности доступа к средствам Java Script, и потому призываю вас тщательно продумывать, какую функциональность приложения вы сможете обеспечить для таких пользователей, но при этом также советую использовать любую возможность для применения шаблонов, поскольку они весьма удобны и практичны.

Первый пример шаблона данных

Наилучший способ изучения шаблонов данных — сразу же взяться за дело. Для демонстрации основных возможностей шаблонов мы используем листинг 12.3. Поскольку элементный состав документа самым непосредственным образом связан с кодом шаблона, помещаемым в элемент `script`, то в данном листинге документ представлен в полном виде, ибо так вам будет легче увидеть общую картину. В последующих примерах листинги содержат лишь те фрагменты кода, которые имеют прямое отношение к обсуждаемому вопросу.

Листинг 12.3. Первый пример шаблона данных

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var data = [
                { name: "Астры", product: "astor",
                  stocklevel: "10", price: 2.99},
                { name: "Нарциссы", product: "daffodil",
                  stocklevel: "12", price: 1.99},
                { name: "Розы", product: "rose",
                  stocklevel: "2", price: 4.99},
                { name: "Пионы", product: "peony",
                  stocklevel: "0", price: 1.50},
                { name: "Примулы", product: "primula",
                  stocklevel: "1", price: 3.12},
                { name: "Подснежники", product: "snowdrop",
                  stocklevel: "15", price: 0.99},
            ];

            $('#flowerTmpl').tmpl(data).appendTo('#row1');
```

```
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                value="0" required />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2"class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

В последующих разделах мы разобьем пример на отдельные части и проанализируем код каждой из них по отдельности. Когда данные являются частью документа, они называются *встроенными данными* (inline data). Альтернативой им являются *дистанционные данные* (remote data), хранящиеся на сервере отдельно от документа. Мы рассмотрим дистанционные данные несколько позже, а пока что можно заметить, что этот вопрос тесно связан с поддержкой Ajax, которую предоставляет библиотека jQuery, что является темой глав 14 и 15.

Определение данных

Пример начинается с определения данных. В нашем случае данные — это массив объектов, каждый из которых описывает отдельный вид цветочной продукции. Соответствующий фрагмент кода приведен в листинге 12.4.

Листинг 12.4. Определение данных

```
var data = [
    { name: "Астры", product: "astor",
      stocklevel: "10", price: 2.99},
    { name: "Нарциссы", product: "daffodil",
      stocklevel: "12", price: 1.99},
    { name: "Розы", product: "rose",
      stocklevel: "2", price: 4.99},
    { name: "Пионы", product: "peony",
      stocklevel: "0", price: 1.50},
    { name: "Примулы", product: "primula",
      stocklevel: "1", price: 3.12},
```

```
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];
...
```

Данные выражаются в виде одного или нескольких объектов JavaScript. Библиотека шаблонов jQuery предоставляет значительную гибкость в выборе объектов, которые могут быть использованы в качестве данных, но представленный выше формат, соответствующий формату данных JSON (см. главу 14), является наиболее распространенным.

Совет. Формат JSON играет очень важную роль, поскольку его часто используют при работе с Ajax, о чем пойдет речь в главах 14 и 15.

В этом примере массив состоит из шести объектов, каждый из которых имеет ряд свойств, описывающих конкретный продукт: отображаемое имя, имя продукта, имеющееся количество единиц товара и цена.

Определение шаблона

Как вы, наверное, и сами догадываетесь, центральным элементом библиотеки шаблонов является *шаблон данных* (data template). Он представляет собой набор HTML-элементов, содержащих заполнители, которые соответствуют различным свойствам объектов данных. Шаблон для этого примера показан в листинге 12.5.

Листинг 12.5. Определение шаблона данных

```
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" data-price="${price}"
            data-stock="${stocklevel}"
            value="0" required />
    </div>
</script>
...
```

Первое, на что следует обратить внимание, — это то, что шаблон помещается в элемент `script`, атрибуту `type` которого присваивается значение несуществующего типа — `text/x-jquery-tmpl`. Это сделано для того, чтобы браузер не пытался интерпретировать содержимое шаблона как обычную HTML-разметку. Хотя это и несущественно, но такой практики следует придерживаться, поскольку она чрезвычайно полезна и позволит вам избежать множества потенциальных проблем в будущем.

Второй момент, на котором я хочу заострить ваше внимание, — это то, что для присвоения имени шаблону, определенному в элементе `script`, используется атрибут `id`. В данном случае именем шаблона служит `flowerTmpl`. Чтобы применить к данным шаблон, необходимо знать его имя.

Содержимое шаблона будет применено ко всем объектам в массиве данных, что приведет к созданию набора HTML-элементов для каждого объекта. Вы видите, что структура шаблона в целом соответствует набору элементов, которые использовались в предыдущих главах для представления различных видов цветочной продук-

ции. Главное, чем они отличаются, — это элементы кода, выделенные в листинге полужирным шрифтом и выполняющие функции *заполнителей* (data placeholders).

В процессе обработки шаблона вместо каждого заполнителя подставляется значение свойства, взятое из текущего объекта. Например, для первого объекта массива вместо заполнителя `${product}` будет подставлено значение свойства `product`, т.е. `astor`. Таким образом, часть шаблона

```
<img src="${product}.png"/>
```

преобразуется в следующий HTML-фрагмент:

```
<img src="astor.png"/>
```

Подстановка значений — не единственное, что могут делать шаблоны. Другие их возможности обсуждаются далее.

Применение шаблона

Для объединения шаблона с данными используется метод `tmpl()`. При этом вы указываете данные, которые должны использоваться, и применяемый к ним шаблон. Пример использования этого метода приведен в листинге 12.6.

Листинг 12.6. Применение шаблона данных

```
...
$('#flowerTmpl').tmpl(data).appendTo('#row1');
...
```

Здесь мы выбираем элемент, который содержит шаблон, используя для этой цели функцию `$()`, и вызываем для полученного результата метод `tmpl()`, передавая ему в качестве аргумента данные, которые хотим обработать.

Метод `tmpl()` возвращает стандартный объект `jQuery`, который содержит элементы, полученные из шаблона. В данном случае это приводит к набору элементов `div`, каждый из которых содержит элементы `img`, `label` и `input`, сконфигурированные для одного из объектов, содержащихся в массиве данных. Для вставки всего набора в качестве дочернего элемента в элемент `row1` используется метод `appendTo()`. Результат представлен на рис. 12.2.

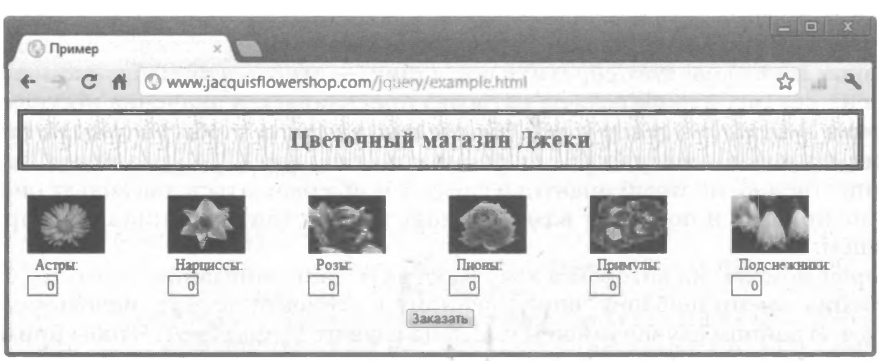


Рис. 12.2. Использование шаблонов данных

Модификация результата

Полученный результат не совсем нас устраивает, поскольку все элементы, соответствующие различным цветам, отображаются в одном ряду. Но поскольку мы имеем дело с объектом jQuery, расположить элементы так, как нам надо, не составит большого труда. В листинге 12.7 показано, как это можно сделать, воздействуя на результат работы метода tmpl().

Листинг 12.7. Обработка результата, возвращаемого шаблоном

```
    <script type="text/javascript">
        $(document).ready(function() {

            var data = [
                { name: "Астры", product: "astor",
                  stocklevel: "10", price: 2.99},
                { name: "Нарциссы", product: "daffodil",
                  stocklevel: "12", price: 1.99},
                { name: "Розы", product: "rose",
                  stocklevel: "2", price: 4.99},
                { name: "Пионы", product: "peony",
                  stocklevel: "0", price: 1.50},
                { name: "Примулы", product: "primula",
                  stocklevel: "1", price: 3.12},
                { name: "Подснежники", product: "snowdrop",
                  stocklevel: "15", price: 0.99},
            ];

            $('#flowerTmpl').tmpl(data)
                .slice(0, 3).appendTo('#row1').end().end()
                .slice(3).appendTo('#row2');

        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                value="0" required />
        </div>
    </script>
...
```

В этом примере методы slice() и end() используются для сужения и расширения набора выбранных элементов, а метод appendTo() — для добавления поднаборов элементов, сгенерированных с помощью шаблона, в различные ряды.

Обратите внимание: для возврата набора в исходное состояние, в котором он находился до применения методов slice() и appendTo(), метод end() пришлось вызывать два раза подряд. Ничего противозаконного в этом нет, и я охотно использую метод end(), чтобы выполнить необходимые действия в рамках одной инструкции, но последовательность end().end() не вызывает у меня восторга. В подобных случаях я предпочитаю разбивать всю последовательность действий на ряд отдельных операций, как показано в листинге 12.8.

Листинг 12.8. Разбиение набора элементов с использованием нескольких инструкций

```
...
var templResult = $('#flowerTmpl').tmpl(data);
templResult.slice(0, 3).appendTo('#row1');
templResult.slice(3).appendTo("#row2");
...
```

В обоих случаях результат будет одним и тем же: представление совокупности продуктов двумя рядами, в каждом из которых отображается по три вида цветов, как показано на рис. 12.3.

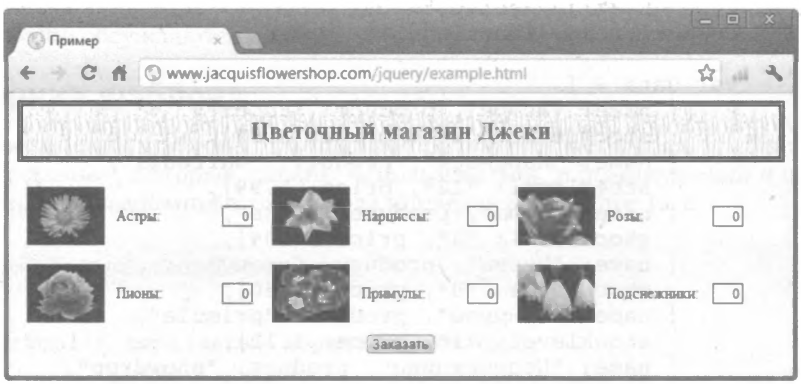


Рис. 12.3. Корректировка результата для получения требуемого макета страницы

Изменение способа предоставления входных данных

Другой возможный подход заключается в изменении способа передачи данных методу `tmpl()`. Соответствующий пример приведен в листинге 12.9.

Листинг 12.9. Корректировка выходных результатов путем изменения способа передачи входных данных шаблону

```
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];
```

```
            var template = $('#flowerTmpl');
            template.tmpl(data.slice(0, 3)).appendTo("#row1");
            template.tmpl(data.slice(3)).appendTo("#row2");

        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                value="0" required />
        </div>
    </script>
...
```

В этом сценарии распределение элементов по рядам осуществляется путем двукратного использования шаблона — по одному разу для каждого ряда. Соответствующая часть объектов данных каждый раз передается шаблону с помощью метода `slice()`. Несмотря на отличие данного подхода от предыдущего, мы получаем тот же результат, который был представлен на рис. 12.3.

Вычисление выражений

Объекты данных можно использовать не только для получения значений свойств. Если поместить между двумя фигурными скобками выражение JavaScript, то движок шаблонов вычислит его и вставит в сгенерированную шаблоном HTML-разметку. Соответствующий пример приведен в листинге 12.10.

Листинг 12.10. Вычисление выражений в шаблоне

```
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                value="${stocklevel > 0 ? 1 : 0}" required />
        </div>
    </script>
...
```

В этом шаблоне значение атрибута `value` элемента `input` устанавливается на основании значения свойства `stocklevel` с помощью тернарного условного оператора. Выражение, заключенное в фигурные скобки, играет ту же роль, какую играло бы записанное вместо него непосредственное значение свойства. Если значение свойства `stocklevel` больше нуля, то значение `value` устанавливается равным 1, в противном случае — 0. Вид полученной страницы в окне браузера представлен на рис. 12.4. Значение `stocklevel`, большее нуля, установлено для всех цветов, кроме пионов.

Рассмотренный пример иллюстрирует основную схему работы с шаблонами: данные объединяются с шаблоном для получения DOM-объектов, которые затем добавляются в документ с использованием основной функциональности jQuery.

Для генерации содержимого можно использовать как непосредственно заданные значения, так и вычисляемые выражения.

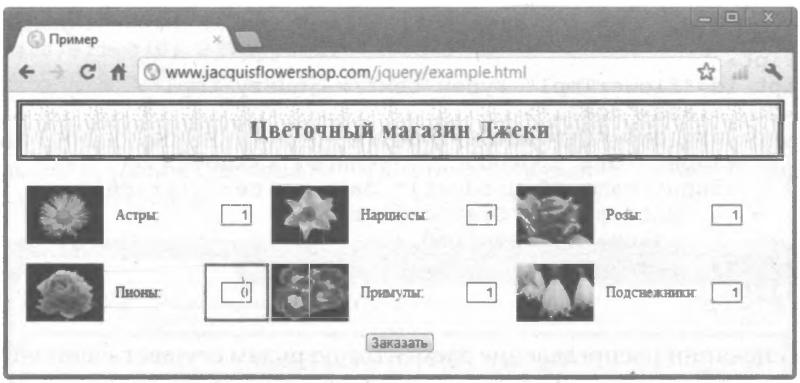


Рис. 12.4. Вычисление выражений в шаблоне

Использование переменных шаблона

Шаблоны не являются сценариями JavaScript. Любое содержимое, которое вы добавляете в элемент `script`, считается частью шаблона и будет включаться в выходной результат. Чтобы сделать шаблоны более гибкими, вам предоставляется небольшое количество контекстных переменных, которые можно использовать в дескрипторах заполнителей. Краткое описание этих переменных содержится в табл. 12.2, а их подробному рассмотрению посвящены следующие разделы.

Таблица 12.2. Контекстные переменные шаблона

Переменная	*Описание*
`$data`	Возвращает текущий элемент данных
`$item`	Возвращает текущий экземпляр шаблона
`$`	Функция `$()` библиотеки jQuery

Использование переменной `$data`

Переменная `$data` возвращает текущий элемент данных, к которому применяется шаблон. Например, используемая в этой главе переменная `$data` будет поочередно ссылаться на каждый из объектов, соответствующих отдельным видам цветов. Для получения данных в предыдущем листинге в шаблоне использовался тернарный условный оператор. Такой подход вполне допустим, однако чтение получаемых при этом шаблонов часто вызывает затруднения, чего, разумеется, желательно не допускать.

Я всегда стараюсь уменьшить объем кода шаблона до необходимого минимума и поэтому предпочитаю использовать переменную `$data` внутри функций JavaScript, которые затем вызываю из шаблона. Соответствующий демонстрационный пример приведен в листинге 12.11.

Листинг 12.11. Использование переменной `$data` в шаблоне

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var template = $('#flowerTmpl');
        template.tmpl(data.slice(0, 3)).appendTo("#row1");
        template.tmpl(data.slice(3)).appendTo("#row2");
    });

    function stockDisplay(product) {
        return product.stocklevel > 0 ? 1 : 0;
    }

</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" data-price="${price}"
            data-stock="${stocklevel}"
            value="${stockDisplay($data)}" required />
    </div>
</script>
...
```

В этом примере определяется функция `stockDisplay()`, возвращающая значение, которое должно отображаться в элементе `input`. Аргументом этой функции является объект данных, который мы получаем внутри шаблона с использованием переменной `$data`. Учитывая, что речь идет всего лишь о простом тернарном операторе, разница в удобочитаемости кода по сравнению с предыдущим вариантом не очень значительна, но в случае более сложных выражений или в случае многократного использования выражения в пределах одного шаблона она будет гораздо более ощутимой.

Предупреждение. Определяя функции, которые будут вызываться из шаблона, будьте внимательны. Дело в том, что такие функции должны определяться до вызова метода `tmpl()`. Я всегда стараюсь помещать их в конце элемента `script`, но если функция должна находиться внутри обработчика события `ready`, то непременно следует убеждаться в том, что она была ранее определена. Другой распространенной ошибкой является то, что функцию часто определяют *внутри* шаблона.

Использование функции $() внутри шаблона

В применяемых внутри шаблона заполнителях можно использовать функцию $() библиотеки jQuery, однако при этом очень важно не забывать, что элементы, генерируемые посредством шаблона, не присоединяются к документу и поэтому не будут попадать в наборы выбранных элементов jQuery. Я редко использую указанную возможность, поскольку обычно меня в большей степени интересуют элементы и связанные с ними данные, которые я генерирую самостоятельно. Тем не менее соответствующий демонстрационный пример приведен в листинге 12.2.

Листинг 12.12. Использование функции $() в шаблоне

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<h2>Специальное предложение на сегодняшний день:
            <span id=offer data-discount='0.50'>"
            + "скидка 50 центов</span></h2>")
            .insertAfter('h1')
            .css({ color: "red", fontSize: "14pt",
            textAlign: "center" });

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var template = $('#flowerTmpl');
        template.tmpl(data.slice(0, 3)).appendTo("#row1");
        template.tmpl(data.slice(3)).appendTo("#row2");
    });

    function stockDisplay(product) {
        return product.stocklevel > 0 ? 1 : 0;
    }

</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}"
            data-price="${price - $('#offer').data('discount')}"
            data-stock="${stocklevel}"
```

```
            value="${stockDisplay($data)}" required />
    </div>
</script>
...
```

В этом примере в документ добавлен элемент `h2`, содержащий элемент `span`, атрибут `data-discount` которого определяет размер предоставляемой скидки. В шаблоне метод `$()` используется для нахождения элемента `span`, а метод `data()` — для получения значения его атрибута (о методе `data()` говорится в главе 8). Когда шаблон используется для генерации элементов, цена, указанная в каждом из объектов данных, уменьшается на размер скидки.

Я включил этот пример исключительно для полноты описания общей картины, поскольку многое в нем мне самому не нравится. Во-первых, при таком способе использования функции `$()` мы вынуждены выполнять поиск элемента `span` в документе для каждого из обрабатываемых объектов данных, что влечет за собой дополнительные накладные расходы. Во-вторых, в шаблон добавляется код, чего следует всячески избегать. В-третьих, перемещение обработки данных в функцию позволило бы абстрагировать способ, который используется для определения цены на основании шаблона. Как бы то ни было, можно использовать обращения к функции `$()` внутри шаблонов, и не исключено, что такой подход отлично впишется в ваш стиль программирования.

Использование переменной `$item`

Объект, возвращаемый переменной `$item`, решает несколько задач, которые мы последовательно рассмотрим в данной главе. Первая из них — обеспечение возможности обмена дополнительными данными между сценарием JavaScript и шаблоном. Соответствующий пример приведен в листинге 12.13. О других применениях этого объекта мы поговорим далее.

Листинг 12.13. Передача параметров шаблону с помощью переменной `$item`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<h2>Специальное предложение на сегодняшний день:
            <span id=offer data-discount='0.50'>"
            + "скидка 50 центов</span></h2>")
            .insertAfter('h1')
            .css({ color: "red", fontSize: "14pt",
            textAlign: "center" });

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
```

```
            stocklevel: "15", price: 0.99},
        ];

        var options = {
            discount: $('#offer').data('discount'),
            stockDisplay: function(product) {
                return product.stocklevel > 0 ? 1 : 0;
            }
        };
        var template = $('#flowerTmpl');
        template.tmpl(data.slice(0, 3), options)
            .appendTo("#row1");
        template.tmpl(data.slice(3), options)
            .appendTo("#row2");
    });
</script>

<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}"
            data-price="${price - $item.discount}"
            data-stock="${stocklevel}"
            value="${$item.stockDisplay($data)}" required />
    </div>
</script>
...
```

В этом примере мы создаем объект `options`, для которого определяются свойство (`discount`) и метод (`stockDisplay()`). Затем этот объект передается методу `tmpl()` в качестве второго аргумента. Доступ к свойствам и методам объекта из шаблона обеспечивает переменная `$item`, поэтому для обращения к значению свойства `discount` используется следующий код:

```
${price - $item.discount}
```

Для вызова функции `stockDisplay()` применяется такой код:

```
$item.stockDisplay($data)
```

Как видите, для обработки скидки в цене, значение которой мы ранее получали с помощью функции `$()`, здесь используется свойство `discount` объекта `options`. В результате мы не только улучшили стиль кода, но и добились того, что поиск элемента `span` в документе приходится выполнять всего лишь один раз.

Совет. Хочу обратить ваше внимание на необходимость включения префикса `$` в имена контекстных переменных: `$item` и `$data`. Такой же префикс обязателен и в конструкции дескриптора шаблона `${...}`, используемой для подстановки значений в шаблон. Пропуск любого из этих префиксов является одной из наиболее распространенных ошибок.

Использование вложенных шаблонов

При создании сложных приложений иногда имеет смысл разбить большой шаблон на несколько частей, объединение которых происходит уже на стадии выполнения приложения. Как будет показано далее, такой способ объединения шаблонов

обеспечивает более гибкое управление выводом. Мы начнем с самого элементарного. В листинге 12.14 показано, каким образом один шаблон может ссылаться на другой.

Листинг 12.14. Вложенные шаблоны

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<h2>Специальное предложение на сегодняшний день:
            <span id=offer data-discount='0.50'>"
            + "скидка 50 центов</span></h2>")
            .insertAfter('h1')
            .css({ color: "red", fontSize: "14pt",
            textAlign: "center" });

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var options = {
            discount: $('#offer').data('discount'),
            stockDisplay: function(product) {
                return product.stocklevel > 0 ? 1 : 0;
            }
        };
        var template = $('#flowerTmpl');
        template.tmpl(data.slice(0, 3), options)
            .appendTo("#row1");
        template.tmpl(data.slice(3), options)
            .appendTo("#row2");
    });

</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        {{tmpl($data, $item) "#inputTmpl"}}
    </div>
</script>

<script id="inputTmpl" type="text/x-jquery-tmpl">
    <input name="${product}"
        data-price="${price - $item.discount}"
        data-stock="${stocklevel}"
```

```
        value="${$item.stockDisplay($data)}" required />
</script>
...
```

В этом примере шаблон разбит на две части. Первая из них, шаблон `flowerTmpl`, вызывается для каждого элемента массива данных. В свою очередь, этот шаблон вызывает шаблон `inputTmpl` для создания элементов `input`. Вызов второго шаблона осуществляется с помощью дескриптора `{{tmpl}}`:

```
{{tmpl($data, $item) "#inputTmpl"}}
```

В этом вызове используются три аргумента. Первые два — это текущий элемент данных и объект `options`; эти аргументы заключаются в круглые скобки. Третий аргумент — это вызываемый шаблон. Его можно задавать либо jQuery-селектором (что и сделано выше), либо переменной или функцией, определенной в сценарии.

Использование вложенных шаблонов с массивами

При передаче вложенному шаблону одиночного значения или объекта указанный шаблон применяется только один раз, как в предыдущем примере. Если же вложенному шаблону передается массив объектов, то он генерирует элементы для каждого элемента массива, как показано в листинге 12.15.

Листинг 12.15. Использование вложенных шаблонов в случае массивов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            {
            rowid: "row1",
            flowers:
                [{ name: "Астры", product: "astor",
                   stocklevel: "10", price: 2.99},
                { name: "Нарциссы", product: "daffodil",
                  stocklevel: "12", price: 1.99},
                { name: "Розы", product: "rose",
                  stocklevel: "2", price: 4.99}]
            },
            {
            rowid: "row2",
            flowers:
                [{ name: "Пионы", product: "peony",
                   stocklevel: "0", price: 1.50},
                { name: "Примулы", product: "primula",
                  stocklevel: "1", price: 3.12},
                { name: "Подснежники", product: "snowdrop",
                  stocklevel: "15", price: 0.99}]
            }
        ];

        $('div.drow').remove();
        $('#rowTmpl').tmpl(data).appendTo('div.dtable');
    });
```

```
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" data-price="${price}"
            data-stock="${stocklevel}"
            value="${stocklevel}" required />
    </div>
</script>
<script id="rowTmpl" type="text/x-jquery-tmpl">
    <div id="${rowid}" class="drow">
        {{tmpl($data.flowers) '#flowerTmpl'}}
    </div>
</script>
...
```

Чтобы продемонстрировать эту возможность, я превратил объект данных в массив из двух объектов. В каждом из этих объектов определяются ключ `rowid`, используемый в качестве идентификатора ряда, и массив из трех объектов, представляющих различные виды цветочной продукции.

Далее я удаляю из документа элементы `div`, соответствующие рядам, и использую шаблон `rowTmpl` для обработки данных. Этот шаблон генерирует новые элементы `div` вместо прежних для организации табличной компоновки страницы с помощью стилей CSS, а обработка массива данных, описывающих цветы, осуществляется с помощью дескриптора `{{tmpl}}`, как показано ниже.

```
<script id="rowTmpl" type="text/x-jquery-tmpl">
    <div id="${rowid}" class="drow">
        {{tmpl($data.flowers) '#flowerTmpl'}}
    </div>
</script>
```

Несмотря на то что шаблон вызывается лишь один раз, механизм шаблонов использует его, чтобы сгенерировать элементы для каждого элемента массива, что в конечном счете обеспечивает создание элементов на уровне ячеек таблицы, соответствующих отдельным видам цветов.

Тот факт, что входные данные заранее структурируются таким образом, чтобы соответствовать требуемой компоновке страницы, может вызвать у вас вполне обоснованные возражения против применения такого подхода. В связи с этим в листинге 12.16 показано, как получить тот же результат путем соответствующего разбиения входных данных с одновременным преобразованием их исходного формата (данные редко поступают в формате, идеально приспособленном для их непосредственного отображения).

Листинг 12.16. Преобразование встроенных данных для использования вложенных шаблонов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var OriginalData = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
```

```
            stocklevel: "12", price: 1.99},
        { name: "Розы", product: "rose",
          stocklevel: "2", price: 4.99},
        { name: "Пионы", product: "peony",
          stocklevel: "0", price: 1.50},
        { name: "Примулы", product: "primula",
          stocklevel: "1", price: 3.12},
        { name: "Подснежники", product: "snowdrop",
          stocklevel: "15", price: 0.99},
    ];

    var itemsPerRow = 4;
    var slicedData = [];

    for (var i = 0, j = 0; i < originalData.length;
        i+= itemsPerRow, j++) {
        slicedData.push({
            rowid: "row" + j,
            flowers: originalData
                .slice(i, i + itemsPerRow)
        });
    }

    $('div.drow').remove();
    $('#rowTmpl').tmpl(sliceData).appendTo('div.dtable');
});

</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" data-price="${price}"
            data-stock="${stocklevel}"
            value="${stocklevel}" required />
    </div>
</script>
<script id="rowTmpl" type="text/x-jquery-tmpl">
    <div id="${rowid}" class="drow">
        {{tmpl($data.flowers) '#flowerTmpl'}}
    </div>
</script>
...
```

В этом примере элементы распределяются по рядам с использованием цикла `for`, и методу `tmpl()` передается объект, содержащий отформатированные данные. Число элементов в ряду определяется значением переменной `itemsPerRow`, которое в данном случае установлено равным 4. Результат представлен на рис. 12.5.

В некотором смысле мы здесь просто переносим сложности из одного места документа в другое. Мы вынуждены либо предварительно форматировать данные, либо допустить использование более сложного шаблона. Избежать этого, по-видимому, невозможно, поскольку ожидать большей помощи от механизма шаблонов не приходится. Выбор оптимального баланса между сложностью кода и сложностью шаблона зависит от формата данных, с которыми вы работаете. Я рекомендую всегда начинать с экспериментов, испытывая различные подходы, пока не

будет найден вариант, который устроит вас и будет понятен другим и возможность поддержки которого в будущем не вызывает у вас никаких сомнений.

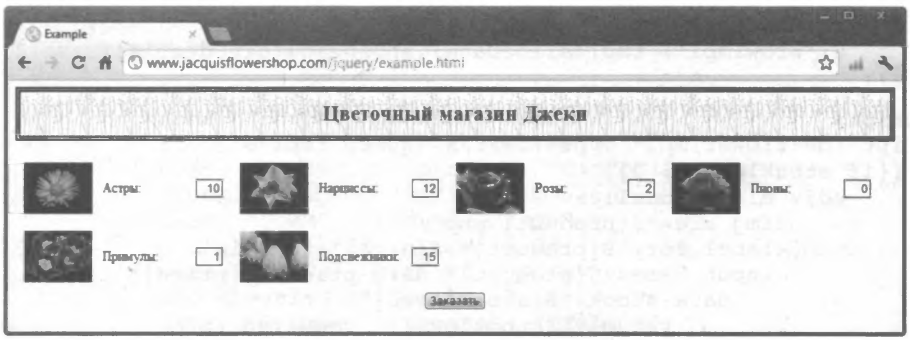


Рис. 12.5. Приведение формата данных в соответствие с шаблонами

Использование условных шаблонов

Механизм шаблонов обеспечивает возможность динамического принятия решений относительно использования различных частей шаблона в зависимости от выполнения определенных условий. Для этого существуют дескрипторы {{if}} и {{/if}}, пример использования которых представлен в листинге 12.17.

Листинг 12.17. Применение различных частей шаблона в зависимости от условий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var OriginalData = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var itemsPerRow = 3;
        var slicedData = [];

        for (var i = 0, j = 0; i < originalData.length;
            i+= itemsPerRow, j++) {
            slicedData.push({
                rowid: "row" + j,
                flowers: originalData
```

```
                    .slice(i, i + itemsPerRow)
            });
        }

        $('div.drow').remove();
        $('#rowTmpl').tmpl(sliceData).appendTo('div.dtable');
    });

</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    {{if stocklevel > 0}}
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                    value="${stocklevel}" required />
        </div>
    {{/if}}
</script>
<script id="rowTmpl" type="text/x-jquery-tmpl">
    <div id="${rowid}" class="drow">
        {{tmpl($data.flowers) '#flowerTmpl'}}
    </div>
</script>
...
```

Условие указывается в дескрипторе `{{if}}`, и часть шаблона, заключенная между этим дескриптором и дескриптором `{{/if}}`, будет использоваться, только если результат вычисления условного выражения окажется равным `true`. Если же этот результат равен `false`, то указанная часть шаблона игнорируется. В данном случае проверяется значение свойства `stocklevel`, и если оно равно нулю, то игнорируется весь шаблон `flowerTmpl`. Это означает, что отображаться будут лишь те продукты, которые имеются в наличии на складе, как показано на рис. 12.6.

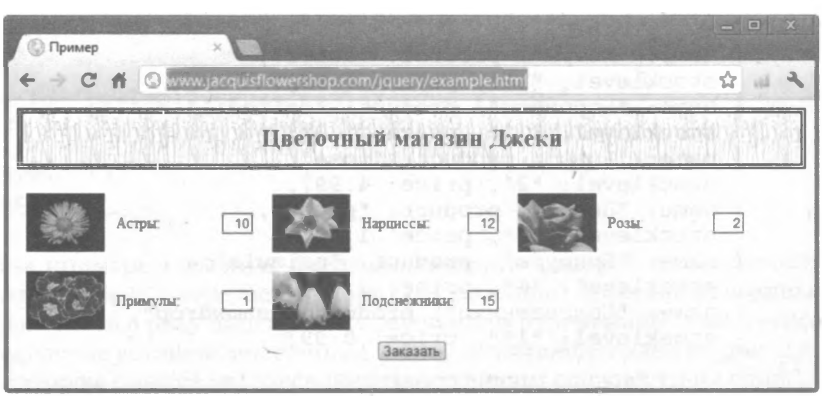


Рис. 12.6. Рендеринг шаблона лишь для тех видов цветочной продукции, которые имеются в наличии

Более сложные условия можно задавать с помощью дескриптора `{{else}}`, позволяющего определить часть шаблона, которая должна использоваться в тех случаях, когда результатом вычисления выражения в дескрипторе `{{if}}` является `false`. Соответствующий пример приведен в листинге 12.18.

Листинг 12.18. Использование дескриптора {{else}}

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var OriginalData = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var itemsPerRow = 3;
        var slicedData = [];

        for (var i = 0, j = 0; i < originalData.length;
            i+= itemsPerRow, j++) {
            slicedData.push({
                rowid: "row" + j,
                flowers: originalData
                    .slice(i, i + itemsPerRow)
            });
        }

        $('div.drow').remove();
        $('#rowTmpl').tmpl(sliceData).appendTo('div.dtable');
    });

</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    {{if stocklevel > 0}}
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                    value="${stocklevel}" required />
        </div>
    {{else}}
         <div class="dcell">
            <img src="${product}.png" style="opacity:0.5"/>
            <span style="color: grey">${name} (Нет в наличии)
            </span>
         </div>
    {{/if}}
</script>
<script id="rowTmpl" type="text/x-jquery-tmpl">
    <div id="${rowid}" class="drow">
        {{tmpl($data.flowers) '#flowerTmpl'}}
```

```
    </div>
</script>
...
```

В этом примере для тех видов продукции, которые имеются на складе, отображается один набор элементов, а для тех, которые отсутствуют, — другой. Можно пойти еще дальше и поместить условие в дескриптор {{else}}, что создаст условную конструкцию, эквивалентную конструкции else...if, как показано в листинге 12.19.

Листинг 12.19. Применение условия в дескрипторе {{else}}

```
...
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    {{if stocklevel > 5}}
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                    value="${stocklevel}" required />
        </div>
    {{else stocklevel > 0}}
        <div class="dcell">
            <img src="${product}.png"/>
            <label style="color:red" for="${product}">
                ${name}: (Небольшое количество)</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}"
                    value="${stocklevel}" required />
        </div>
    {{else}}
        <div class="dcell">
            <img src="${product}.png" style="opacity:0.5"/>
            <span style="color: grey">
                ${name} (Нет в наличии)
            </span>
        </div>
    {{/if}}
</script>
...
```

В этом сценарии отображаемый набор элементов определяется тем, сколько единиц товарной продукции того или иного вида имеется на складе: больше пяти, пять, меньше пяти или вообще нет в наличии. Я ввел лишь незначительные различия между элементами, генерируемыми для каждого из этих условий, но вы вправе применять эти возможности для генерации совершенно другого содержимого. Наконец, в случае необходимости можно вызывать по условию другие шаблоны. Результат работы приведенного выше сценария представлен на рис. 12.7.

Управление обработкой массивов

Для управления способом обработки массивов данных в шаблоне используется дескриптор {{each}}. Этот подход представляет собой альтернативу использованию вызовов вложенных шаблонов, которое мы ранее обсуждали. Пример использования дескриптора {{each}} приведен в листинге 12.20.

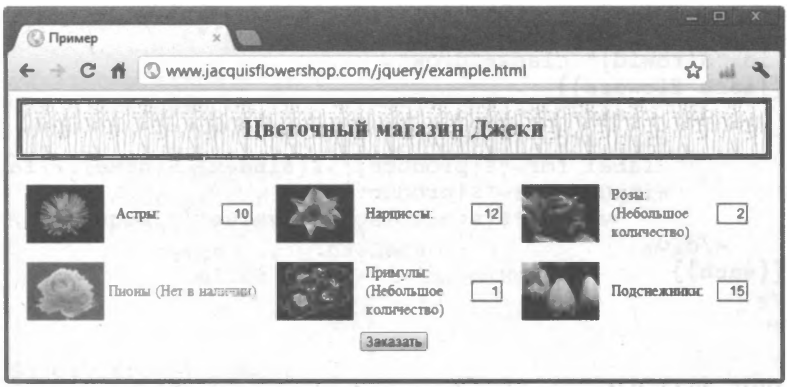


Рис. 12.7. Использование условных операторов в шаблоне

Листинг 12.20. Использование дескриптора шаблона {{each}}

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var OriginalData = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var itemsPerRow = 3;
        var slicedData = [];

        for (var i = 0, j = 0; i < originalData.length;
            i+= itemsPerRow, j++) {
            slicedData.push({
                rowid: "row" + j,
                flowers: originalData
                    .slice(i, i + itemsPerRow)
            });
        }

        $('div.drow').remove();
        $('#flowerTmpl').tmpl(sliceData).appendTo('div.dtable');
    });

    function stockDisplay(product) {
        return product.stocklevel > 0 ? 1 : 0;
    }
```

```
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div id="${rowid}" class="drow">
        {{each flowers}}
            <div class="dcell">
                <img src="${product}.png"/>
                <label for="${product}">${$index} ${name}:</label>
                <input name="${product}"
                    value="${stockDisplay($value)}" required />
            </div>
        {{each}}
    </div>
</script>
...
```

Рендеринг содержимого, заключенного между дескрипторами {{each}} и {{/each}}, выполняется для каждого элемента указанного массива, причем внутри дескриптора {{each}} можно ссылаться на отдельные свойства так, как это обычно делается. Для получения текущего элемента массива используется переменная $value, а для получения индекса текущего элемента — переменная $index.

Совет. Имена переменных $index и $value заданы по умолчанию. Если по каким-то причинам оказывается желательным использование других имен, их следует указать в качестве аргументов в дескрипторе {{each}}. Например, если вместо переменной $index вы хотите использовать переменную $i, а вместо переменной $value — $v, то необходимо записать следующее:

```
{{each($i, $v) flowers}}
```

Поэлементная обработка результата вычисления выражения

Если предоставить дескриптору {{each}} выражение, то рендеринг всего содержимого вплоть до дескриптора {{/each}} будет выполняться для каждого элемента, входящего в результат вычисления выражения. Пример использования этой методики представлен в листинге 12.21, где набор элементов данных фильтруется с целью удаления тех из них, которые имеются лишь в небольшом количестве или вообще отсутствуют.

Листинг 12.21. Использование выражения в дескрипторе {{each}}

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var OriginalData = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
```

```
            stocklevel: "15", price: 0.99},
        ];

        var itemsPerRow = 3;
        var slicedData = [];

        for (var i = 0, j = 0; i < originalData.length;
            i+= itemsPerRow, j++) {
            slicedData.push({
                rowid: "row" + j,
                flowers: originalData
                    .slice(i, i + itemsPerRow)
            });
        }

        $('div.drow').remove();
        $('#flowerTmpl').tmpl(sliceData).appendTo('div.dtable');
    });

    function stockDisplay(product) {
        return product.stocklevel > 0 ? 1 : 0;
    }
    function filterLowStock(flowers) {
        var result = [];
        for (var i = 0; i < flowers.length; i++) {
            if (flowers[i].stocklevel > 2) {
                result.push(flowers[i]) }
        }
        return result;
    }
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div id="${rowid}" class="drow">
        {{each filterLowStock(flowers)}}
            <div class="dcell">
                <img src="${product}.png"/>
                <label for="${product}">${name}:</label>
                <input name="${product}"
                    value="${stockDisplay($value)}" required />
            </div>
        {{each}}
    </div>
</script>
...
```

Здесь внутри дескриптора {{each}} вызывается функция `filterLowStock()`, ограничивающая количество итерируемых элементов. Результат представлен на рис. 12.8.

Отключение HTML-кодирования

По умолчанию механизм шаблонов кодирует значения, которые вы вставляете в шаблон, для их безопасного отображения на странице. Под этим подразумевается, что, например, символы < или <, имеющие в HTML особое значение, заменяются или кодируются таким образом, чтобы они не интерпретировались как часть разметки, обозначающей HTML-элемент. Обычно подобные преобразования полезны,

но если вы имеете дело с данными, которые содержат HTML-разметку, то могут возникнуть проблемы. Соответствующий пример приведен в листинге 12.22.

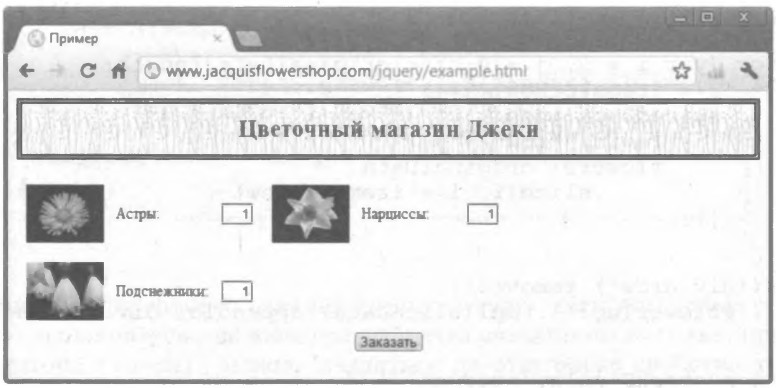


Рис. 12.8. Использование выражения в дескрипторе {{each}}

Листинг 12.22. Работа с данными, содержащими HTML-разметку

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              elem: "<img src=astor.png/>},
            { name: "Нарциссы", product: "daffodil",
              elem: "<img src=daffodil.png/>},
            { name: "Розы", product: "rose",
              elem: "<img src=rose.png/>},
            { name: "Пионы", product: "peony",
              elem: "<img src=peony.png/>},
            { name: "Примулы", product: "primula",
              elem: "<img src=primula.png/>},
            { name: "Подснежники", product: "snowdrop",
              elem: "<img src=snowdrop.png/>},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");
    });
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        ${elem}
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" required />
    </div>
</script>
...
```

В этом примере каждый элемент данных содержит свойство, значение которого представляет собой HTML-элемент, выводящий изображение продукта. В шаблоне для отображения этого свойства используется ссылка на имя свойства — `${elem}`. Суть проблемы становится понятной при рассмотрении рис. 12.9.

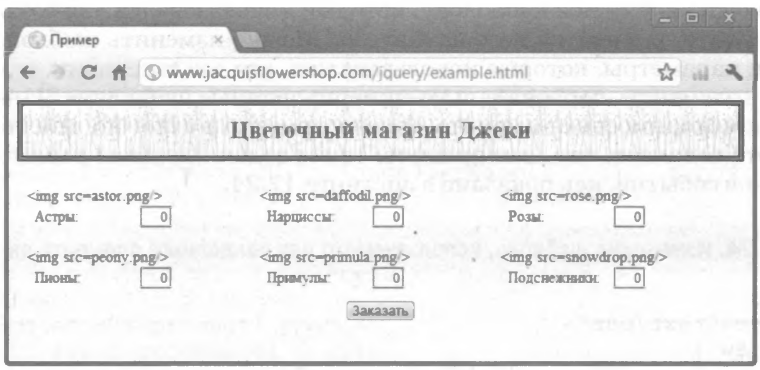


Рис. 12.9. Последствия кодирования HTML-содержимого объекта данных

Этой проблемы можно избежать, используя дескриптор `{{html}}`. Он сообщает механизму шаблонов о том, что содержимое, с которым вы работаете, должно отображаться "как есть". Пример применения этого дескриптора показан в листинге 12.23.

Предупреждение. Не позволяйте себе безоглядно использовать этот дескриптор и никогда не доверяйте содержимому, переданному пользователями. В противном случае можно очень легко вставить в страницу вредоносное содержимое, в том числе и сценарии, которые могут подменить обработчики событий в коде.

Я вовсе не шучу. Обращение с данными, вставляемыми в шаблон с помощью этого дескриптора, требует большой осторожности. Такие данные могут быть чрезвычайно опасными, если только вы не можете гарантировать их абсолютную надежность. Не забывайте о том, что вредоносные изменения могут быть внесены в код вашими же коллегами, особенно из числа оперативного персонала, среди которых всегда найдется кто-то, кому вы чем-то не угодили. Возможно, именно в данный момент он как раз готовит против вас очередные козни. Не доверяйте никому!

Листинг 12.23. Использование дескриптора `{{html}}`

```
...
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        {{html elem}}
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" required />
    </div>
</script>
...
```

Используйте дескриптор `{{html}}` вместе со значением, которое вы хотите вставить в шаблон, и тогда последовательность экранирующих символов будет опущена.

Манипулирование шаблонами из обработчиков событий

Всегда можно вернуться к шаблону, который использовался для создания какого-либо элемента, и внести в него изменения. Можно изменить шаблон, изменить данные или параметры, которые использовались для его генерации, и даже модифицировать элементы, первоначально сгенерированные шаблоном. Чаще всего это диктуется необходимостью изменить часть документа в ответ на действия пользователя, а это означает, что, как правило, такие задачи должны решаться внутри обработчиков событий, как показано в листинге 12.24.

Листинг 12.24. Изменение шаблона, используемого для рендеринга элемента данных

```
...
<style type="text/css">
    .bigview {
        border: medium solid black;
        position: relative;
        top: -10px;
        left: -10px;
        background-color: white;
    }
    .bigview > img {
        width: 160px;
        height: 120px;
    }
</style>
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('div.cell').mouseenter(handleMouse);

        function handleMouse(e) {
            var tmplItem = $.tmplItem(this);
            var template = e.type == "mouseenter" ?
                "#flowerTmplSel" : "#flowerTmpl";
```

```
            tmplItem.tmpl = $(template).template();
            tmplItem.update();
            $('div.dcell').unbind()
                .bind(e.type == "mouseenter" ?
                    "mouseleave" : "mouseenter", handleMouse);
        }
    });
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" required />
    </div>
</script>
<script id="flowerTmplSel" type="text/x-jquery-tmpl">
    <div class="dcell bigview">
        <img src="${product}.png"/>
        {{if $data.stocklevel > 0}}
            В наличии: ${stocklevel}  Цена: $${price}
        {{else}}
            (Нет в наличии)
        {{/if}}
    </div>
</script>
...
```

Этот пример нуждается в подробных объяснениях. Здесь есть что обсуждать, а тех, кто недостаточно подготовлен, могут подстерегать потенциальные ловушки. Основная суть того, что делает сценарий, довольно проста: имеется встроенный объект данных, рендеринг которого осуществляется с помощью шаблона `flowerTmpl`.

По завершении рендеринга содержимого мы используем метод `mouseenter()` для регистрации функции `handleMouse()` в качестве обработчика событий для элементов, соответствующих селектору `div.dcell`. Таковыми являются элементы верхнего уровня, рендеринг которых выполняет шаблон.

При наведении указателя мыши на один из этих элементов `div` выполняется функция `handleMouse()`. Первое, что делает эта функция, — получает экземпляр шаблона, связанный с элементом, который запустил событие.

```
var tmplItem = $.tmplItem(this);
```

Это объект того же рода, который можно было бы получить с помощью переменной `$item` внутри шаблона (о чем уже говорилось ранее), однако использование объекта приносит гораздо больше пользы. Свойства и методы объекта экземпляра шаблона перечислены в табл. 12.3.

Таблица 12.3. Свойства объекта экземпляра шаблона

Имя	*Описание*
`nodes`	Набор объектов `HTMLElement`, сгенерированных шаблоном. Дерево объектов сохраняется, поэтому данное свойство возвращает лишь элементы верхнего уровня
`data`	Объект данных, используемый для рендеринга элемента
`tmpl`	Шаблон, используемый для генерации выбранного элемента
`parent`	Родительский шаблон, если элемент был сгенерирован из вложенного шаблона

Окончание табл. 12.3

Имя	Описание
<свойства>	Свойства объекта `options` (если он используется) непосредственно доступны в данном объекте экземпляра шаблона

Вернемся к примеру. Мы выбираем шаблон, который хотим использовать для отображения выбранного элемента, исходя из типа события. Если указатель мыши находится над элементом, то выбирается шаблон `flowerTmplSel`.

```
var template = e.type == "mouseenter" ?
    "#flowerTmplSel" : "#flowerTmpl";
```

Далее мы используем свойство `tmpl` экземпляра шаблона для связывания шаблона с элементом данных.

```
tmplItem.tmpl = $(template).template();
```

Анализируя то, что здесь происходит, полезно знать, что механизм шаблонов запоминает, каким образом он генерировал содержимое, которое вы пытаетесь изменить. Он сохраняет информацию о том, какой объект данных *и* какой шаблон были использованы. Эта методика чрезвычайно удобна и работает даже в случае использования условий и вложенных шаблонов. С помощью свойства `tmpl` мы сообщаем механизму шаблонов о том, что хотим генерировать содержимое для элемента данных с помощью другого шаблона, который использовался первоначально для генерации элемента, запустившего обрабатываемое событие (вполне возможно, что для полного понимания смысла только что прочитанного вам придется перечитать это предложение несколько раз).

Закончив внесение изменений, мы вызываем метод `update()` для экземпляра шаблона, чтобы выполнить повторный рендеринг содержимого с использованием новых значений (в данном случае — нового шаблона).

```
tmplItem.update();
```

Результат применения нового шаблона представлен на рис. 12.10.

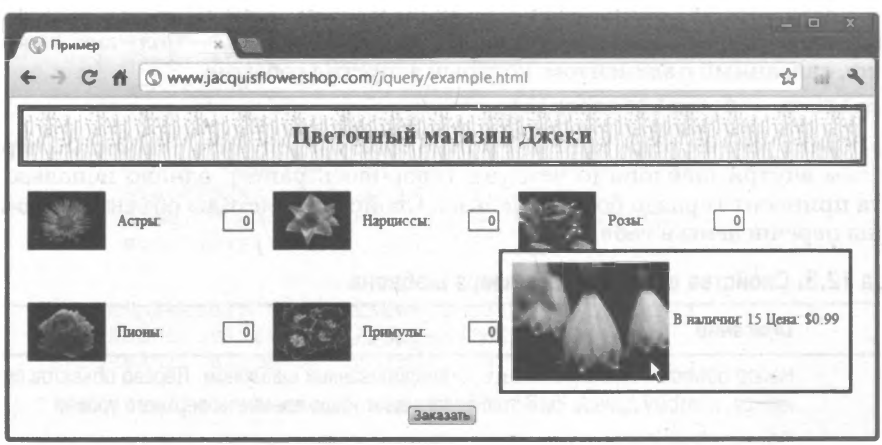


Рис. 12.10. Использование шаблона, отличного от первоначального, при наведении указателя мыши на элемент

Последняя инструкция функции `handleMouse()` имеет следующий вид:

```
$('div. dcell').unbind() .bind(e.type == "mouseenter" ?
    "mouseleave" : "mouseenter", handleMouse);
```

При замене шаблона сгенерированные ранее элементы удаляются из документа и заменяются теми, которые сгенерировал новый шаблон. Это означает, что одновременно удаляются и связанные с ними обработчики событий. Такой характер взаимодействия библиотеки шаблонов с основной библиотекой jQuery для нас неблагоприятен и, в частности, означает, что метод `live()` (см. главу 9) не сможет корректно работать, так что вы сами должны позаботиться о повторном добавлении нужных обработчиков.

Изменение данных, используемых шаблоном

Можно изменить данные, используемые шаблоном, и в этом случае механизм шаблонов регенерирует элементы с использованием новых значений. Соответствующий пример приведен в листинге 12.25.

Листинг 12.25. Изменение данных, используемых шаблоном

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('<button>Изменить данные</button>')
            .prependTo("#buttonDiv").click(function(e) {
                var item = $.tmplItem($('div.dcell').first());
                item.data.product = "orchid";
                item.data.name = "Орхидеи";
                item.update();
                e.preventDefault();
        })
    });
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
```

```
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" required />
    </div>
</script>
...
```

В данном примере в документ была добавлена кнопка. При щелчке на ней из первого элемента извлекается шаблон, соответствующий селектору `div.dcell`, после чего используется свойство `data` для внесения изменений в объекты данных.

```
var item = $.tmplItem($('div.dcell').first());
item.data.product = "orchid";
item.data.name = "Орхидеи";
```

Далее вызывается метод `update` для повторной генерации контента с использованием новых значений. Полученный эффект можно увидеть на рис. 12.11.

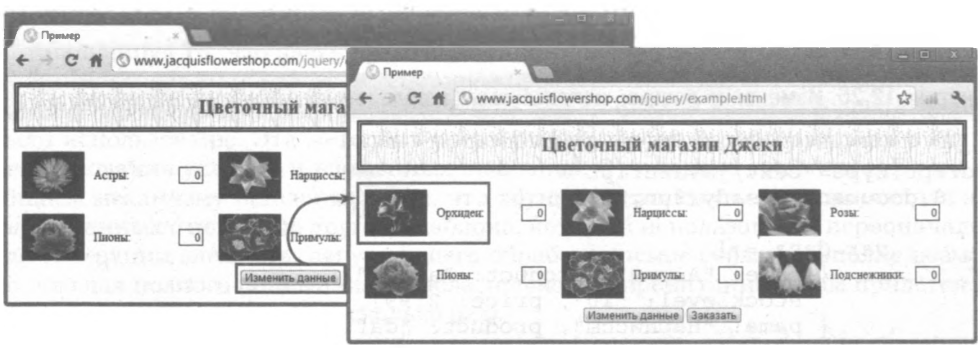


Рис. 12.11. Изменение данных, связанных с шаблоном

Располагая ссылкой на исходный объект данных, можно получить тот же результат, применив изменение непосредственно к данным, как показано в листинге 12.26.

Листинг 12.26. Непосредственное изменение значений объекта данных

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];
```

```
        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

            data[0].name = "Орхидеи";

            data[0].product = "orchid";

        $('<button>Изменить данные</button>')
            .prependTo("#buttonDiv").click(function(e) {
                var item = $.tmplItem($('div.dcell').first())
                    .update();
                e.preventDefault();
        })
    });
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" required />
    </div>
</script>
...
```

Резюме

В этой главе вы познакомились с библиотекой шаблонов jQuery, которая предоставляет отличные возможности для преобразования данных JavaScript в HTML-элементы без написания множества строк запутанного кода. Несмотря на неопределенность статуса этого подключаемого модуля, он необычайно полезен и пользуется популярностью. Учитывая обеспечиваемые им удобства в работе и широкие возможности, я рекомендую вам использовать его в своих проектах, хотя его и нельзя назвать идеальным.

ГЛАВА 13

Работа с формами

В этой главе вы познакомитесь со средствами поддержки HTML-форм, предоставляемыми библиотекой jQuery. Изложенный здесь материал поможет вам освежить свои знания о событиях, связанных с формами, однако большая часть главы посвящена подключаемому модулю, который предоставляет замечательный механизм проверки корректности данных, вводимых пользователем в полях формы, прежде чем они будут отправлены на сервер. Если вам уже приходилось разрабатывать веб-приложения на основе форм, то вы согласитесь, что пользователи склонны вводить в поля формы все что угодно, и поэтому проверка вводимых ими данных играет очень важную роль.

Глава начинается с представления серверного сценария Node.js, который будет использоваться в этой части. В данной главе этот сценарий используется всего лишь для отображения значений, введенных в форму, но в последующих главах он будет играть более серьезную роль. Перечень тем, рассматриваемых в данной главе, приведен в табл. 13.1.

Таблица 13.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Настройка сервера Node.js	Используйте приведенный в этой главе сценарий (доступен на сайте книги)	1, 2
Реагирование на получение или потерю фокуса ввода элементом `form`	Используйте методы `focus()` и `blur()`	3
Реагирование на событие отправки формы пользователем или его прерывание	Используйте метод `change()`	4
Реагирование на изменение значения, введенного пользователем в форму	Используйте метод `change()`	5, 6
Проверка корректности значений, введенных в форму	Используйте подключаемый модуль Validation	7
Настройка подключаемого модуля проверки правильности заполнения форм	Передайте объект отображения данных методу `validate()`	8
Определение и применение правил проверки с помощью класса	Используйте методы `addClassRules()` и `addClass()`	9–12
Применение правил проверки непосредственно к элементам	Используйте метод `rules()`	13, 14
Применение правил проверки с использованием имен элементов	Добавьте свойство `rules` в объект `options`	15
Применение правил проверки с использованием атрибутов элементов	Определите атрибуты, соответствующие отдельным контрольным проверкам	16

Окончание табл. 13.1

Задача	Решение	Листинг
Определение пользовательских сообщений для правил, применяемых с использованием имен и атрибутов элементов	Добавьте свойство `message` в объект `options`, установленный для объекта отображения данных, который определяет пользовательские сообщения	17, 18
Определение пользовательских сообщений для правил, применяемых непосредственно к элементам	Включите объект отображения данных, определяющий сообщения, в качестве аргумента при вызове метода `rules()`	19
Создание пользовательской контрольной проверки	Используйте метод `addMethod()`	20, 21
Форматирование сообщений о ходе проверки	Используйте свойства `highlight`, `unhighlight`, `errorElement` и `errorClass` объекта `options`	22–26
Использование отчета о проверке	Используйте свойства `errorContainer` и `errorLabelContainer`	27

Подготовка сервера Node.js к работе

В этой главе для получения и обработки данных формы, отправляемой браузером, используется сервер Node.js. Мы не будем углубляться в подробное описание принципов работы Node.js, но одной из причин, по которым для книги был выбран именно этот сервер, послужило то, что он полностью основан на JavaScript. Таким образом, разрабатывая серверные сценарии, вы сможете использовать все свои навыки программирования клиентских приложений.

Совет. Если вы намерены выполнять примеры из этой главы, обратитесь к главе 1, в которой подробно описано, где можно взять файлы сервера Node.js и дополнительного модуля, на использование которого я буду опираться. Для загрузки серверного сценария и всей необходимой HTML-документации посетите сайт Apress.com.

Серверный сценарий, который мы будем использовать в данной главе, приведен в листинге 13.1. Я представляю его вам как "черный ящик" и описываю лишь входные и выходные данные.

Листинг 13.1. Сценарий `formserver.js` для сервера Node.js

```
var http = require('http');
var querystring = require('querystring');

http.createServer(function (req, res) {
    console.log("[200 OK] " + req.method + " to " + req.url);

    if (req.method == 'POST') {
        var dataObj = new Object();
        var contentType = req.headers["content-type"];
        var fullBody = '';

        if (contentType) {
            if (contentType
             .indexOf("application/x-www-form-urlencoded") > -1) {
             req.on('data', function(chunk) { fullBody +=
```

```
                chunk.toString();});
               req.on('end', function() {
                res.writeHead(200, "OK", {'Content-Type':
                 'text/html; charset=UTF-8'});
                res.write('<html><head><title>POST-данные
                 </title></head><body>');
                res.write('<style>th, td {text-align:left;
                 padding:5px; color:black}\n');
                res.write('th {background-color:grey; color:white;
                 min-width:10em}\n');
                res.write('td {background-color:lightgrey}\n');
                res.write('caption {font-weight:bold}</style>');
                res.write('<table border="1">
                 <caption>Данные формы</caption>');
                res.write('<tr><th>Имя</th><th>Значение</th>');
                var dBody = querystring.parse(fullBody);
                for (var prop in dBody) {
                res.write("<tr><td>" + prop + "</td><td>" +
                 dBody[prop] + "</td></tr>");
                }
                res.write('</table></body></html>');
                res.end();
               });
              }
        }
    }
}).listen(9999);
console.log("Ready on port 9999");
```

Чтобы выполнить этот сценарий, необходимо ввести в командной строке следующую команду.

```
node.exe formserver.js
```

В другой операционной системе, отличной от той, которую использую я, команда будет выглядеть иначе. За более подробными сведениями обратитесь к документации Node.js. Чтобы продемонстрировать функциональность Node.js, используемую в данной главе, воспользуемся образцом документа, приведенным в листинге 13.2.

Листинг 13.2. Пример документа для этой главы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var data = [
                { name: "Астры", product: "astor",
                  stocklevel: "10", price: 2.99},
                { name: "Нарциссы", product: "daffodil",
                  stocklevel: "12", price: 1.99},
                { name: "Розы", product: "rose",
```

```
                stocklevel: "2", price: 4.99},
              { name: "Пионы", product: "peony",
                stocklevel: "0", price: 1.50},
              { name: "Примулы", product: "primula",
                stocklevel: "1", price: 3.12},
              { name: "Подснежники", product: "snowdrop",
                stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

    });
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" required />
    </div>
</script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2"class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Элементы, соответствующие отдельным видам цветочной продукции, генерируются с помощью шаблонов (глава 12). В элемент `form` добавлен атрибут `action`, значение которого говорит о том, что для отправки формы используется следующий URL-адрес:

```
http://node.jacquisflowershop.com/order
```

Я использую два разных сервера. С первым из них (`www.jacquisflowershop.com`) мы работали до сих пор. Он предназначен для доставки статического содержимого: HTML-документов, файлов сценариев и изображений. На моем компьютере эти функции выполняет сервер Microsoft IIS 7.5, но можно использовать для этого любой другой сервер, который вас устраивает (я выбрал IIS, поскольку многие из моих книг посвящены технологиям компании Microsoft и этот сервер уже был у меня установлен и готов к работе).

Второй сервер (`node.jacquisflowershop.com`) — это сервер Node.js (использующий приведенный выше сценарий), и именно ему будут пересылаться данные при отправке формы с веб-страницы нашего образца документа. В этой главе нас не

будет интересовать, что именно происходит с полученными данными на сервере. Все наше внимание будет сосредоточено на форме. На рис. 13.1 показан документ, в котором элементы `input` содержат некоторые значения.

Щелчок на кнопке Заказать приводит к отправке формы на сервер `Node.js`, который отправляет ответную информацию браузеру, как показано на рис. 13.2.

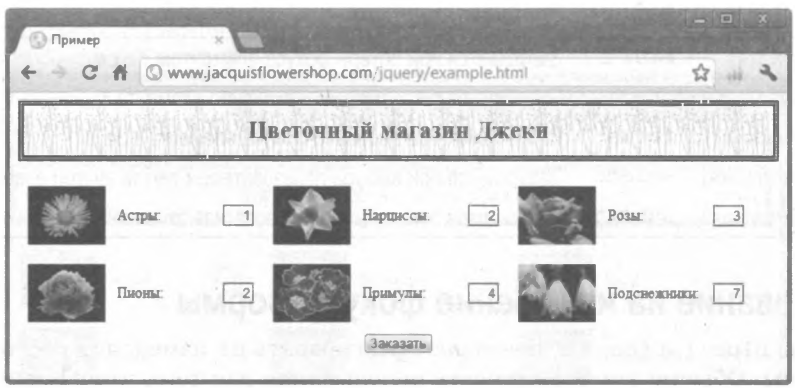


Рис. 13.1. Ввод данных в элементы `input`

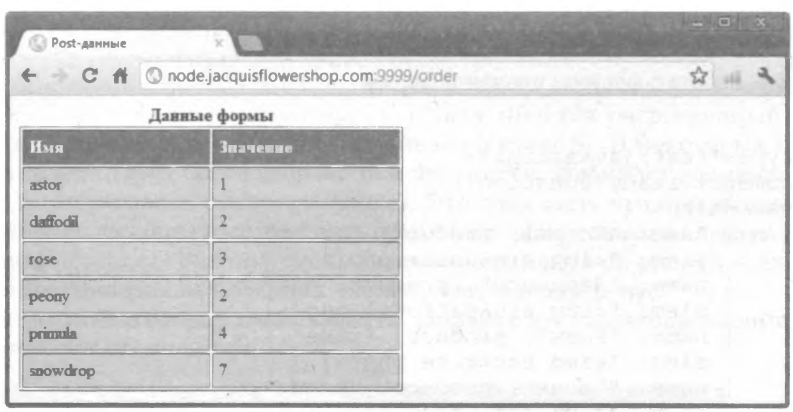


Рис. 13.2. Ответ сервера Node.js

Я понимаю, что ничего особенно интересного в этом ответе нет, однако форму куда-то нужно отправлять, а я не хочу уходить в сторону от нашей основной темы, отвлекаясь на рассмотрение вопросов, связанных с разработкой серверов.

Повторение методов, связанных с обработкой событий формы

Как говорилось в главе 9, jQuery предоставляет ряд методов, предназначенных для обработки событий, связанных с формой. Сейчас имеет смысл освежить ваши знания в этой области, поскольку формы являются предметом нашего рассмотрения. Интересующие нас методы и события, которые им соответствуют, перечислены в табл. 13.2.

Совет. Не забывайте о том, что в jQuery определен набор расширенных селекторов, которые выбирают элементы формы. Более подробные сведения по этой теме содержатся в главе 5.

Таблица 13.2. Методы jQuery для обработки событий формы

Метод	*Событие*	*Описание*
`blur(`*функция*`)`	`blur`	Запускается при потере фокуса элементом `form`
`change(`*функция*`)`	`change`	Запускается при изменении значения элемента `form`
`focus(`*функция*`)`	`focus`	Запускается при передаче фокуса элементу `form`
`select(`*функция*`)`	`select`	Запускается при выборе пользователем текста внутри элемента `form`
`submit(`*функция*`)`	`submit`	Запускается при попытке пользователя отправить форму

Реагирование на изменение фокуса формы

Методы `blur()` и `focus()` позволяют реагировать на изменения состояния фокуса формы. Обычно эта возможность используется для того, чтобы помочь пользователю сориентироваться в том, какой элемент имеет фокус (и, таким образом, какой элемент будет принимать вводимые данные). Соответствующий пример приведен в листинге 13.3.

Листинг 13.3. Работа с фокусом элемента формы

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = [
            { name: "Астры", product: "astor",
              elem: "<img src=astor.png/>},
            { name: "Нарциссы", product: "daffodil",
              elem: "<img src=daffodil.png/>},
            { name: "Розы", product: "rose",
              elem: "<img src=rose.png/>},
            { name: "Пионы", product: "peony",
              elem: "<img src=peony.png/>},
            { name: "Примулы", product: "primula",
              elem: "<img src=primula.png/>},
            { name: "Подснежники", product: "snowdrop",
              elem: "<img src=snowdrop.png/>},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('input').focus(handleFormFocus).blur(handleFormFocus);

        function handleFormFocus(e) {
            var borderVal = e.type == "focus" ?
                "medium solid green" : "";
            $(this).css("border", borderVal);
        }
```

```
    });
</script>
...
```

В этом примере мы выбираем все элементы `input` и регистрируем функцию `handleFormFocus()` в качестве обработчика обоих событий — `focus` и `blur`. Эта функция выделяет элемент зеленой рамкой, когда он принимает фокус, и убирает рамку, когда фокус теряется. Результат представлен на рис. 13.3.

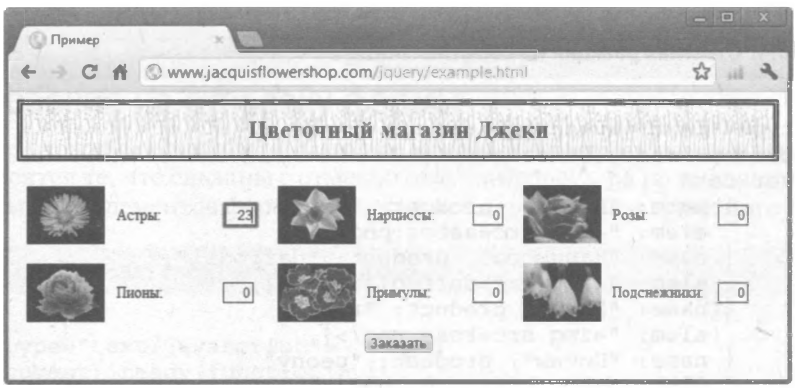


Рис. 13.3. Выделение элемента, получившего фокус

Обратите внимание на то, что здесь использован селектор `input`. Иными словами, элементы выбираются по дескриптору. В jQuery имеется расширенный селектор `:input` (о расширенных селекторах говорилось в главе 5). В некоторых браузерах этот селектор действует более широко и, в частности, выбирает элементы `button`, способные инициировать отправку формы. Это означает, что при использовании расширенного селектора граница будет применяться не только к элементам `input`, но и к элементу `button`. Разницу легко заметить, взглянув на рис. 13.4, на котором для сравнения показана выделенная кнопка, получившая фокус.

Какой из этих селекторов использовать, решаете вы сами, важно лишь, чтобы вы знали, чем они отличаются.

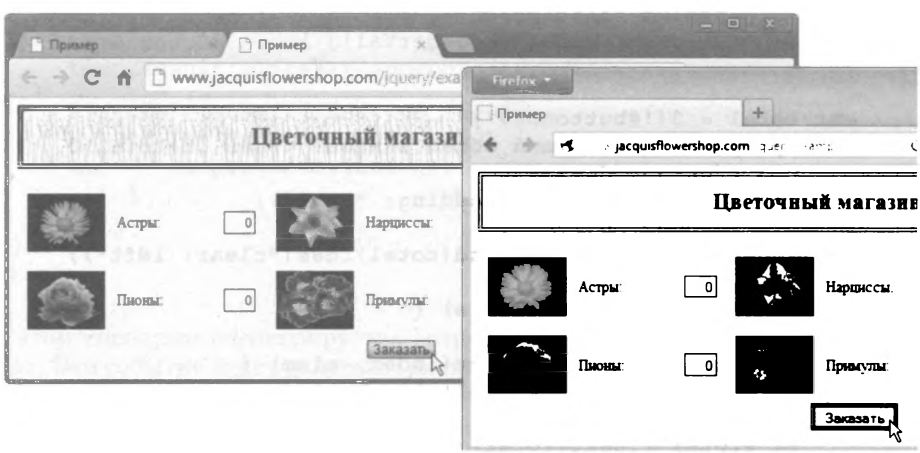


Рис. 13.4. Разница в действии селекторов `input` и `:input`

Реагирование на изменение значений формы

Событие change запускается, когда пользователь изменяет значение элемента form. Это событие особенно полезно, если вы предоставляете совокупную информацию, основанную на значениях, введенных в форме. В листинге 13.4 показано, как это событие можно использовать в документе, представляющем сайт цветочного магазина, для отслеживания общего объема заказа. Здесь используется тот же подход, который применялся при рефакторинге примера в конце части II.

Листинг 13.4. Ответная реакция на событие change

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = [
            { name: "Астры", product: "astor",
              elem: "<img src=astor.png/>},
            { name: "Нарциссы", product: "daffodil",
              elem: "<img src=daffodil.png/>},
            { name: "Розы", product: "rose",
              elem: "<img src=rose.png/>},
            { name: "Пионы", product: "peony",
              elem: "<img src=peony.png/>},
            { name: "Примулы", product: "primula",
              elem: "<img src=primula.png/>},
            { name: "Подснежники", product: "snowdrop",
              elem: "<img src=snowdrop.png/>},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('input').focus(handleFormFocus).blur(handleFormFocus);

        function handleFormFocus(e) {
            var borderVal = e.type == "focus" ?
                "medium solid green" : "";
            $(this).css("border", borderVal);
        }

        var total = $('#buttonDiv')
            .prepend("<div>Общий объем заказа: <span id=total>0
                </span></div>")
            .css({clear: "both", padding: "5px"});
        $('<div id=bbox />')
            .appendTo("body").append(total).css("clear: left");

        $('input').change(function(e) {
            var total = 0;
            $('input').each(function(index, elem) {
                total += Number($(elem).val());
            });
            $('#total').text(total);
        });
```

```
    });
</script>
...
```

В этом примере в ответ на событие change вычисляется сумма всех значений, введенных в полях input, и результат отображается в элементе span, который перед этим был добавлен в документ.

Совет. Обратите внимание на то, что для получения значений элементов input используется метод val().

Реагирование на отправку формы

К более сложным операциям, которые можно выполнять по отношению к формам, относятся те, что связаны с отменой действий браузера по умолчанию, предусмотренных для элементов формы. Простой пример приведен в листинге 13.5.

Листинг 13.5. Перехват события отправки формы

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = [
            { name: "Астры", product: "astor",
              elem: "<img src=astor.png/>},
            { name: "Нарциссы", product: "daffodil",
              elem: "<img src=daffodil.png/>},
            { name: "Розы", product: "rose",
              elem: "<img src=rose.png/>},
            { name: "Пионы", product: "peony",
              elem: "<img src=peony.png/>},
            { name: "Примулы", product: "primula",
              elem: "<img src=primula.png/>},
            { name: "Подснежники", product: "snowdrop",
              elem: "<img src=snowdrop.png/>},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').submit(function (e) {
            if ($('input').val() == 0) {
                e.preventDefault();
            }
        });
    });
</script>
...
```

В этом сценарии регистрируется встроенная функция для обработки события submit. Это событие будет запускаться при выполнении пользователем щелчка на кнопке Заказать. Если значение первого элемента input равно 0, вызывается метод, preventDefault(), который прервет предусмотренное для формы действие по умолчанию, заключающееся в отправке данных на сервер. При любом другом значении отправка формы выполняется.

Совет. Тот же результат можно получить, установив возвращаемое функцией значение равным 0.

Существуют способы отправки формы программным путем. Для этого можно использовать либо функцию jQuery `submit()`, либо метод `submit`, который определен для элементов формы спецификацией HTML5. Пример использования обоих подходов приведен в листинге 13.6.

Листинг 13.6. Явная отправка формы

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = [
            { name: "Астры", product: "astor",
              elem: "<img src=astor.png/>},
            { name: "Нарциссы", product: "daffodil",
              elem: "<img src=daffodil.png/>},
            { name: "Розы", product: "rose",
              elem: "<img src=rose.png/>},
            { name: "Пионы", product: "peony",
              elem: "<img src=peony.png/>},
            { name: "Примулы", product: "primula",
              elem: "<img src=primula.png/>},
            { name: "Подснежники", product: "snowdrop",
              elem: "<img src=snowdrop.png/>},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').submit(function (e) {
            if ($('input').val() == 0) {
                e.preventDefault();
            }
        });

        $('<button>DOM API</button>').appendTo('#buttonDiv')
            .click(function (e) {
            document.getElementsByTagName("form")[0].submit();
            e.preventDefault();
        });

        $('<button>Метод jQuery</button>').appendTo('#buttonDiv')
            .click(function (e) {
            $('form').submit();
            e.preventDefault();
        });
    });
</script>
...
```

Здесь в документ добавлены две кнопки. Щелчок на той из них, которая использует метод jQuery `submit()`, приведет к вызову вашей функции-обработчика, и если первый элемент `input` имеет нулевое значение, то форма не отправляется. В то же время после щелчка на кнопке, которая использует DOM API и вызывает метод

submit(), определяемый элементом формы, ваш обработчик событий будет обойден, и форма будет отправляться в любом случае.

Совет. Разумеется, я рекомендую стремиться использовать методы jQuery, но если вы будете использовать методы DOM, то по крайней мере вам будут понятны получаемые результаты.

Проверка данных формы

Основной причиной, по которой приходится прерывать процесс отправки браузером данных формы на сервер является наше желание убедиться в том, что введенные пользователем в полях формы данные корректны. В определенный момент каждый веб-программист начинает осознавать, что пользователь может ввести в элемент input все, что угодно. Число различных значений, обработка которых может понадобиться, бесконечно, однако, как показывает мой опыт, существуют всего лишь несколько причин, по которым пользователи иногда вводят в форму совершенно неожиданные данные.

Первая из них — это непонимание пользователем того, ввод каких данных от него ожидается. Вы, например, запрашиваете имя, указанное на кредитной карте, а пользователь вводит ее номер.

Вторая причина — это нежелание пользователя предоставлять вам запрошенную информацию, и он лишь стремится поскорее закончить заполнение формы. Он готов ввести любое значение, чтобы перейти к следующему шагу. Если у вас накопилось изрядное количество новых пользователей, которые в качестве своего адреса электронной почты указали a@a.com, то вы поймете, что это действительно является проблемой.

Третьей причиной может быть то, что пользователь просто не в состоянии предоставить вам запрашиваемую информацию. Разве может житель России корректно ответить на вопрос о том, в каком штате он проживает? В России нет такой единицы административного деления, как штат.

Наконец, пользователь может просто ошибиться, что чаще всего связано с обычными опечатками. Например, я печатаю быстро, но не всегда правильно, и поэтому часто ввожу свою фамилию как *Freman*, а не *Freeman*, пропуская одну букву *e*.

Что касается опечаток, то здесь ничего не поделаешь, но от того, как вы справитесь с тремя остальными причинами, зависит, будет ли работа пользователя с вашим приложением идти, как по маслу, или приложение будет лишь раздражать пользователей.

Я не хочу вдаваться в долгие рассуждения относительно дизайна веб-форм и лишь замечу, что лучше всего подходить к решению этого вопроса с точки зрения потребностей пользователя. И если что-то идет не так, постарайтесь взглянуть на проблему (и на возможные пути ее решения) глазами пользователя. Ваши пользователи не знают о том, каким образом вы создавали систему, и им ничего не известно о том, как организованы ваши бизнес-процессы; их интересует лишь конечный результат. Все будут только счастливы, если вы сосредоточитесь на тех задачах, которые пользователь стремится выполнить, и не будете без нужды наказывать его, если он не предоставит требуемые данные.

В библиотеке jQuery имеются все необходимые средства для создания собственной системы контроля введенных данных, но я рекомендую другой подход. Один из подключаемых модулей jQuery, пользующихся наибольшей популярностью, называется Validation, и уже само это название говорит о том, для чего он предназначен, поскольку *валидация данных* — это процесс проверки введенных данных на

предмет их соответствия определенным правилам, условиям или ограничениям, проще говоря — проверки их корректности.

Предупреждение. В этой главе речь идет о *контроле данных на стороне клиента*. Этот процесс дополняет, но не заменяет контроль данных на стороне сервера, в ходе которого данные проверяются в том виде, в каком они поступают на сервер. Проверка данных на клиентской стороне осуществляется в интересах *пользователей*, поскольку она направлена на то, чтобы избавить пользователей от многократных попыток отправки на сервер неправильных данных, которые затем все равно придется исправлять, теряя драгоценное время. Проверка данных на серверной стороне выполняется в интересах *приложения* и направлена на то, чтобы некорректные данные не могли стать источником всевозможных проблем. Вы обязаны использовать оба вида проверки, поскольку обойти проверку на клиентской стороне не составляет большого труда, и она не гарантирует надежной защиты вашего приложения.

Можете либо загрузить подключаемый модуль Validation, перейдя по адресу `http://bassistance.de/jquery-plugins/jquery-plugin-validation`, либо использовать версию, включенную в файлы примеров для книги (см. главу 1). Пример использования этого модуля представлен в листинге 13.7.

Совет. Для подключаемого модуля Validation предусмотрено множество настраиваемых конфигурационных параметров. В этой главе рассматриваются лишь тех из них, которые используются чаще других и охватывают самый широкий диапазон возможных ситуаций. Если вам не будет их хватать, рекомендую познакомиться с остальными параметрами, описанными в прилагаемом к модулю файле документации.

Листинг 13.7. Использование подключаемого модуля Validation

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery.validate.js" type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        div.errorMsg {color: red}
        .invalidElem {border: medium solid red}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            var data = [
                { name: "Астры", product: "astor",
                  stocklevel: "10", price: 2.99},
                { name: "Нарциссы", product: "daffodil",
                  stocklevel: "12", price: 1.99},
                { name: "Розы", product: "rose",
                  stocklevel: "2", price: 4.99},
                { name: "Пионы", product: "peony",
                  stocklevel: "0", price: 1.50},
                { name: "Примулы", product: "primula",
                  stocklevel: "1", price: 3.12},
                { name: "Подснежники", product: "snowdrop",
```

```
                stocklevel: "15", price: 0.99},
            ];

            var templResult = $('#flowerTmpl').tmpl(data);
            templResult.slice(0, 3).appendTo('#row1');
            templResult.slice(3).appendTo("#row2");

            $('form').validate({
                highlight: function(element, errorClass) {
                    $(element).add($(element).parent())
                    .addClass("invalidElem");
                },
                unhighlight: function(element, errorClass) {
                    $(element).add($(element).parent())
                    .removeClass("invalidElem");
                },
                errorElement: "div",
                errorClass: "errorMsg"
            });

            $.validator.addClassRules({
                flowerValidation: {
                    min: 0
                }
            })

            $('input').addClass("flowerValidation")
            .change(function(e) {
                $('form').validate().element($(e.target));
            })
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" required />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2"class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Примечание. Несмотря на то что в стандарте HTML5 предусмотрена поддержка некоторых базовых средств валидации форм, эти средства весьма просты, а разные браузеры все еще интерпретируют положения спецификации во многом различными способами. До тех пор пока сфера действия, возможности и взаимная согласованность средств HTML5 не будут расширены или улучшены, я рекомендую использовать для проверки данных форм только средства библиотеки jQuery.

Импортирование файла JavaScript

Первое, что вы должны сделать, — это добавить в документ подключаемый модуль Validation, как показано ниже.

```
...
<script src="jquery.validate.js" type="text/javascript"></script>
...
```

Здесь указана версия, предназначенная для отладки, но ничто не мешает использовать минимизированную версию этого файла. Кроме того, ввиду популярности данного модуля, он предлагается для загрузки многими, хотя и не всеми, службами CDN.

Настройка параметров проверки

Следующим шагом является настройка параметров проверки формы. Для этого следует вызвать метод `validate()` для тех элементов формы, которые вы хотите проверить. В качестве аргумента методу `validate()` передается объект отображения данных, содержащий конфигурационные настройки, как показано в листинге 13.8.

Листинг 13.8. Настройка параметров валидации

```
...
$('form').validate({
    highlight: function(element, errorClass) {
        $(element).add($(element).parent())
        .addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).add($(element).parent())
        .removeClass("invalidElem");
    },
    errorElement: "div",
    errorClass: "errorMsg"
});
...
```

Здесь задаются значения четырех параметров (`highlight`, `unhighlight`, `errorElement` и `errorClass`), назначение которых мы обсудим позднее.

Определение правил проверки

Своей гибкостью подключаемый модуль Validation во многом обязан разнообразию предусмотренных в нем способов простого и быстрого определения правил проверки корректности ввода. Существует много способов связывания правил с элементами. Я предпочитаю тот из них, который основан на использовании классов. Вы определяете набор правил и связываете их с классом, а когда форма подвергается проверке, правила применяются лишь к тем элементам, которые принадлежат указанному классу. В примере, представленном в листинге 13.9, создается одно правило.

Листинг 13.9. Определение правила проверки

```
...
$.validator.addClassRules({
    flowerValidation: {
        min: 0
    }
})
...
```

В данном случае создается правило, которое будет применяться ко всем элементам, принадлежащим классу `flowerValidation`. Правило состоит в том, что значение должно быть больше или равно `0`. Данное условие выражено в правиле путем указания контрольной проверки `min`. Это лишь один из многих удобных предопределенных видов контрольной проверки, предоставляемых модулем Validation, и все они будут описаны в данной главе.

Применение правил проверки

Связывание правил с элементами формы достигается путем добавления элементов в класс, указанный на предыдущем шаге. Это позволяет настраивать правила для разных типов элементов. В этом примере все элементы обрабатываются одинаково, и потому все элементы ввода выбираются с помощью jQuery и добавляются в класс `flowerValidation`, как показано в листинге 13.10.

Листинг 13.10. Добавление элементов `input` в класс, связанный с выполнением проверки

```
...
$('input').addClass("flowerValidation")
.change(function(e) {
    $('form').validate().element($(e.target));
})
...
```

Здесь также используется функция, привязанная к событию `change`. Она непосредственно выполняет проверку элемента, значение которого было изменено. Это гарантирует немедленную обратную связь с пользователем в случае исправления им ошибки. Результат работы подключаемого модуля Validation представлен на рис. 13.5. Для получения рисунка я ввел `-1` в поле ввода и щелкнул на кнопке Заказать.

Совет. Текст сообщения, выводимого для пользователя, генерируется модулем проверки. О возможности изменения текста сообщений говорится далее.

Как и в случае организации проверки данных собственными средствами, которую мы до этого обсуждали, пользователь не сможет отправить форму до тех пор, пока не устранит проблему, но на этот раз он видит, какие именно значения неверны, и ему даются рекомендации по устранению ошибки. (Согласен, что это сообщение выглядит довольно абстрактно, но, как будет далее показано, можно изменять текст сообщений по своему усмотрению.)

Использование встроенных проверок

Подключаемый модуль Validation поддерживает большое количество встроенных проверок данных, введенных в полях формы. С одним из них (`min`) вы уже по-

знакомились в предыдущем примере. Полный список встроенных проверок представлен в табл. 13.3.

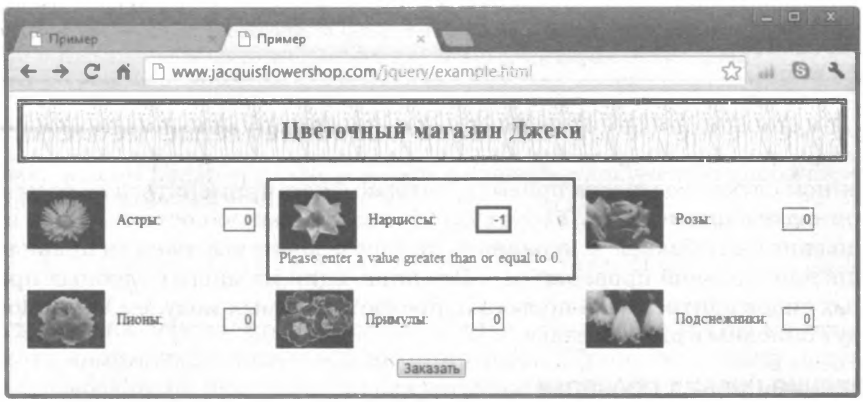


Рис. 13.5. Использование подключаемого модуля Validation

Таблица 13.3. Встроенные проверки, предусмотренные в модуле Validation

Проверка	*Описание*
`creditcard: true`	Значение должно содержать номер кредитной карты
`date: true`	Значение должно быть действительной датой JavaScript
`digits: true`	Значение должно содержать лишь цифры
`email: true`	Значение должно быть действительным адресом электронной почты
`max: maxVal`	Значение не должно превышать `maxVal`
`maxlength: length`	Значение должно содержать не более `length` символов
`min: minVal`	Значение не должно быть меньше `minVal`
`minlength: length`	Значение должно содержать не менее `length` символов
`number: true`	Значение должно быть десятичным числом
`range: [minVal, maxVal]`	Значение должно находиться в пределах указанного диапазона
`rangelength: [minLen, maxLen]`	Значение должно содержать не менее `minLen` и не более `maxLen` символов
`required: true`	Значение обязательно должно быть указано
`url: true`	Значение должно быть URL-адресом

Несколько правил могут быть объединены в одно. Тем самым обеспечиваются компактность и наглядность кода, осуществляющего проверку.

Совет. В дистрибутивный пакет подключаемого модуля Validation входит файл `additional-methods.js`. В нем содержатся дополнительные виды проверок, включая проверку телефонных номеров в форматах, принятых в США и Великобритании, адресов IPv4 и IPv6, а также некоторых дополнительных форматов дат, адресов электронной почты и URL-адресов.

Эти правила могут применяться к элементам несколькими способами. Все они описаны в следующих разделах.

Примечание. Подключаемый модуль Validation поддерживает также *дистанционную валидацию*, при которой проверка данных, вводимых пользователем, выполняется дистанционным сервером. Это может оказаться полезным в тех случаях, когда проверяемые данные не могут передаваться клиенту, поскольку это может быть небезопасно или непрактично (например, при проверке того, не существует ли пользователя с таким именем). Возможности дистанционной проверки данных будут продемонстрированы в главе 16 после того, как в главах 14 и 15 будут представлены средства, на которых она основана.

Применение правил проверки на основании принадлежности классам

Чаще всего я пользуюсь методикой, в которой применение правил проверки основывается на классах элементов. Именно такой подход предпринят в данном примере. Однако ничто не заставляет вас ограничиться только одним видом проверки. Для проверки различных аспектов значения, предоставленного пользователем, можно объединить в одном правиле несколько видов проверки, как показано в листинге 13.11.

Листинг 13.11. Объединение нескольких видов проверки в одном правиле

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $.validator.addClassRules({
```

```
            flowerValidation: {
                required: true,
                digits: true,
                min: 0,
                max: 100
            }
        })

        $('input').addClass("flowerValidation")
        .change(function(e) {
            $('form').validate().element($(e.target));
        })
    });
</script>
...
```

В этом примере проверки `required`, `digits`, `min` и `max` объединены в одно правило, позволяющее убедиться в том, что предоставленное пользователем значение является обязательным для ввода, включает только цифры и находится в интервале от 0 до 100.

Обратите внимание на то, что для связывания правила с классом используется метод `addClassRules()`. Аргументами этого метода являются один или несколько наборов проверок и имя класса, к которому они применяются. Как видно из листинга, метод `addClassRules()` вызывается для свойства `validator` основной функции jQuery `$()`.

Доступ к каждому проверяемому элементу формы осуществляется индивидуально, а это означает, что диагностические сообщения, выводимые для пользователя, будут разными, в зависимости от характера возникшей проблемы, как показано на рис. 13.6.

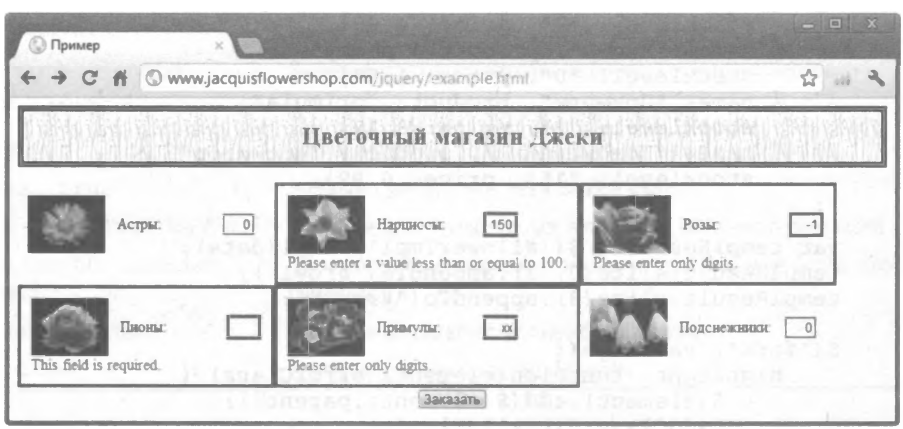


Рис. 13.6. Одновременное применение нескольких видов проверки к элементам формы

Здесь введено несколько значений, каждое из которых не проходит одного из видов проверки. Важно отметить, что проверки выполняются в том порядке, в каком они определены в правиле. Если вы посмотрите на сообщение для продукта `Розы`, то увидите, что оно не прошло проверку `digits`. Изменив порядок определения проверок, вы получите другое сообщение. В листинге 13.12 порядок этих определений изменен.

Листинг 13.12. Изменение порядка применения различных проверок

```
...
$.validator.addClassRules({
    flowerValidation: {
        required: true,
        min: 0,
        max: 100,
        digits: true
    }
})
...
```

В этом примере проверка `digits` передвинута в конец правила. Теперь, если ввести в поле формы значение `-1`, будет выведено сообщение о том, что это значение не проходит проверку `min`, как показано на рис. 13.7.

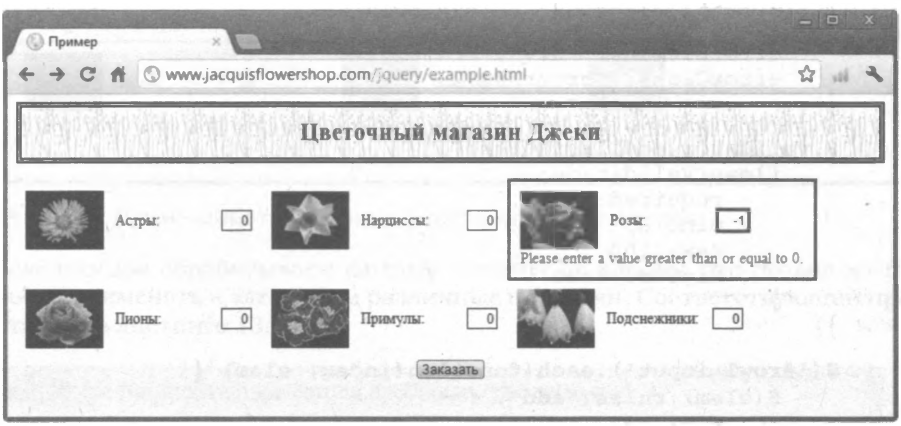


Рис. 13.7. Изменение порядка применения проверок в процессе валидации данных

Применение правил проверки непосредственно к элементам

Следующая методика позволяет применять правила к определенным элементам, как показано в листинге 13.13.

Листинг 13.13. Применение правил проверки к элементам выбранного набора

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
```

```
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $.validator.addClassRules({
            flowerValidation: {
                required: true,
                min: 0,
                max: 100,
                digits: true
            }
        })

        $('#row1 input').each(function(index, elem) {
            $(elem).rules("add", {
                min: 10,
                max: 20
            })
        });

        $('input').addClass("flowerValidation")
            .change(function(e) {
            $('form').validate().element($(e.target));
        })
    });
</script>
...
```

Обратите внимание: мы вызываем метод, определяющий правила, для объекта `jQuery` и передаем ему строку `add` и объект отображения данных с видами проверок, которые хотим выполнить, и их аргументами. Метод `rules()` воздействует лишь на первый элемент выбранного набора, и поэтому для расширения сферы его действия мы должны использовать метод `each()`. В данном случае выбираются все элементы `input`, являющиеся потомками элемента `row1`, к которым и применяются указанные проверки.

Совет. При вызове метода `rules()` можно добавлять и удалять отдельные проверки, используя соответственно методы `add()` и `remove()`.

Правила, применяемые к элементам с использованием методов `rules()`, интерпретируются до того, как будут интерпретироваться правила, применяемые с использованием классов. В контексте нашего примера это означает, что элементы верхнего ряда будут проверяться с использованием значения `min`, равного `10`, и значения `max`, равного `20`, в то время как к другим элементам `input` будут применяться соответственно значения `0` и `100`. Результат представлен на рис. 13.8.

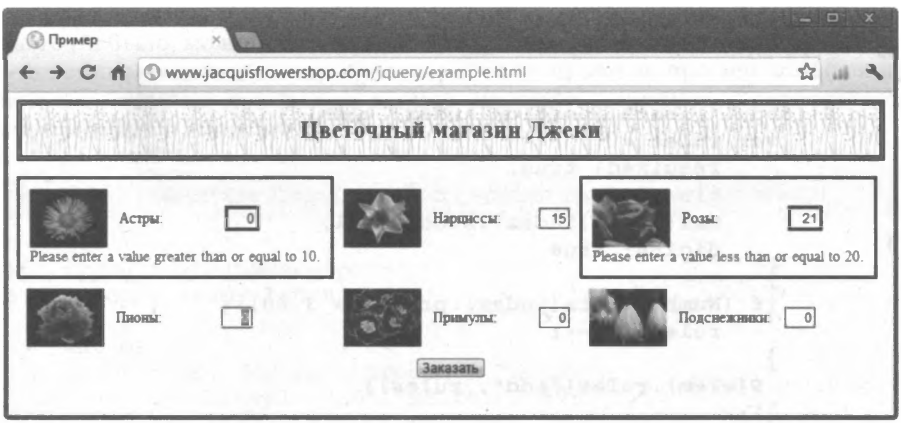


Рис. 13.8. Применение правил непосредственно к элементам

Поскольку мы обрабатываем каждый элемент по одному, это позволяет по отдельности применять к элементам различные проверки. Соответствующий пример представлен в листинге 13.14.

Листинг 13.14. Настройка параметров проверки для элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').validate({
```

```
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $('input').each(function(index, elem) {
            var rules = {
                required: true,
                min: 0,
                max: data[index].stocklevel,
                digits: true
            }
            if (Number(data[index].price) > 3.00) {
                rules.max--;
            }
             $(elem).rules("add", rules);
            });

        $('input').change(function(e) {
            $('form').validate().element($(e.target));
        })
    });
</script>
...
```

В этом примере мы настраиваем значение для `max`, используя объект данных, добавленный в документ для генерации элементов с помощью шаблона. Значение для проверки `max` устанавливается на основании значения свойства `stocklevel` и корректируется в сторону уменьшения, если цена превышает $3. В случае данных, подобных этим, можно выполнять гораздо более полезные проверки. Результат этого изменения представлен на рис. 13.9.

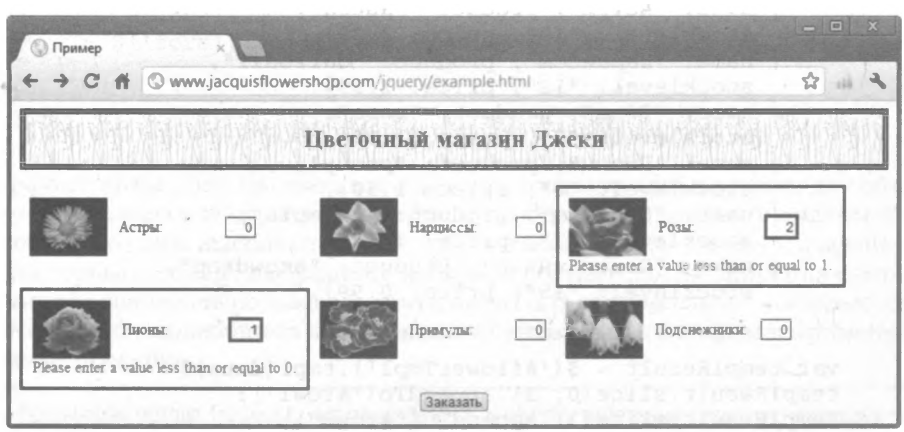


Рис. 13.9. Установка критериев проверки на основании данных

Применение правил проверки на основании значений атрибута name

Правила проверки могут применяться на основании значения атрибута name. В спецификации HTML нет никаких указаний относительно того, что значение атрибута name должно быть уникальным, и этим часто пользуются для выделения целой группы элементов формы в отдельную категорию, присваивая одно и то же значение данного атрибута всем элементам группы. В нашем примере элементы, соответствующие отдельным продуктам, имеют разные значения атрибута name. Как бы то ни было, можно создавать как правила, относящиеся к одному элементу с определенным значением атрибута name, так и правила, применяющиеся ко всем элементам, которым присвоено данное значение атрибута. Соответствующий пример приведен в листинге 13.15.

Листинг 13.15. Применение правил проверки с использованием имен элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        var rulesList = new Object();
        for (var i = 0; i < data.length; i++) {
            rulesList[data[i].product] = {
                min: 0,
                max: data[i].stocklevel,
            }
        }

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
```

```
            errorClass: "errorMsg",
            rules: rulesList
        });

        $('input').change(function(e) {
            $('form').validate().element($(e.target));
        })
    });
</script>
...
```

В этом сценарии правила, основанные на именах элементов, добавляются путем использования свойства `rules` конфигурационного объекта, посредством которого методу `validate()` передаются параметры проверки. Обратите внимание на то, что для создания набора правил используется только объект `data` (свойство `product` элементов которого используется для генерации значений атрибута `name` элементов `input`). Я стараюсь избегать такого подхода, поскольку предпочитаю работать непосредственно с элементами документа, но эта методика может пригодиться, если имеется объект данных и вы хотите настроить параметры проверки еще до того, как элементы формы будут добавляться в документ.

Применение правил проверки на основании значений атрибутов элементов

Существует еще один способ применения правил проверки, основанный на атрибутах и предполагающий использование элементов. Модуль Validation пытается найти элементы с атрибутами, имена которых соответствуют названиям встроенных проверок. Если такие элементы удается найти, то предполагается, что они требуют конкретной проверки. Соответствующий пример приведен в листинге 13.16.

Листинг 13.16. Применение правил проверки с помощью атрибутов элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
```

```
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $('input').change(function(e) {
            $('form').validate().element($(e.target));
        })

    });
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" required
            min="0" max="${stocklevel}"/>
    </div>
</script>
...
```

Мне нравится эта методика, если она используется в сочетании с шаблонами данных, но я считаю, что она загромождает документ, когда применяется к статически определяемым элементам, поскольку сопровождается многократным применением к элементам одних и тех же атрибутов.

Изменение диагностических сообщений проверки

В модуле Validation для всех встроенных проверок определены сообщения об ошибках, используемые по умолчанию, однако все они типовые, и пользователь не всегда может извлечь из них полезную информацию. Это можно пояснить на простом примере. Если установлена проверка `max` с использованием значения `12` в качестве допустимой верхней границы и пользователь вводит в соответствующем поле значение `20`, то сообщение об ошибке будет выглядеть так:

```
Please enter a value less than or equal to 12
```

Это сообщение отражает ограничение, действующее в отношении данного элемента формы, но ничего не говорит пользователю о том, зачем это ограничение необходимо. К счастью, у вас есть возможность изменить текст этих сообщений, вставив в них некоторую дополнительную информацию в соответствии с конкретными задачами. Какой именно метод используется для изменения сообщений об ошибках, зависит в первую очередь от того, каким способом были созданы правила проверки. В тех случаях, когда применяемые правила основаны на классах, изменить сообщения нельзя, но в следующих разделах рассказывается, как определить сообщения в случае использования других методик.

Задание текста диагностических сообщений для проверок на основе атрибутов и имен элементов

Если для связывания правил проверки с элементами используется атрибут `name` или атрибуты, имена которых совпадают с именами встроенных видов проверок,

то текст сообщений можно изменить, добавив свойство `messages` в объект `options`, передаваемый методу `validate()` при настройке параметров проверки. Соответствующий пример приведен в листинге 13.17.

Листинг 13.17. Использование свойства `messages` в объекте `options`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg",
            messages: {
                rose: {
                    max: "У нас нет такого количества роз!"
                },
                peony: {
                    max: "У нас нет такого количества пионов!"
                }
            }
        });

        $('input').change(function(e) {
            $('form').validate().element($(e.target));
        })
    })
</script>
...
```

Нетрудно увидеть, какова структура объекта, предоставляемого в качестве значения свойства message. Вы определяете свойство, используя имя интересующего вас элемента, и задаете его значение в виде объекта отображения, устанавливающего соответствие между названием вида проверки и сообщением об ошибке, которое вы хотите использовать. В данном примере изменяется текст сообщения, относящийся к проверке max для элементов, атрибуты name которых имеют значения rose и peony. Результат представлен на рис. 13.10. Для обоих указанных элементов отображаются измененные диагностические сообщения.

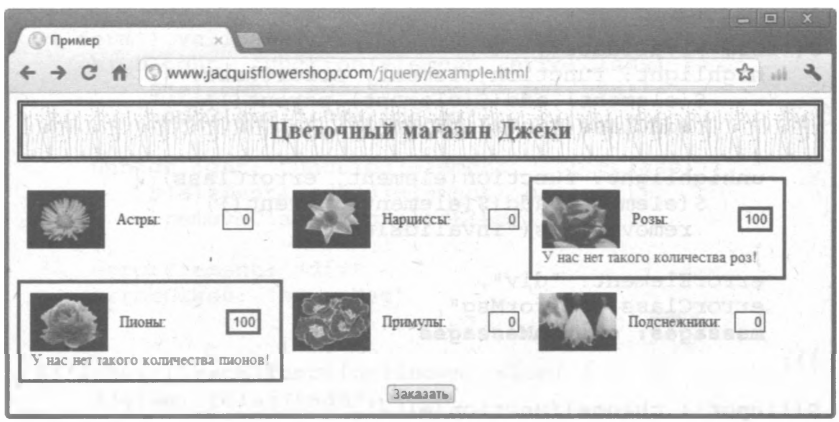


Рис. 13.10. Изменение текста диагностических сообщений с помощью объекта параметров

Значительная часть текста подобных сообщений для различных элементов может повторяться, поэтому обычно я создаю объект, содержащий сообщения, программным путем, как показано в листинге 13.18.

Листинг 13.18. Определение пользовательских диагностических сообщений программным путем

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
```

```
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        var customMessages = new Object();
        for (var i = 0; i < data.length; i++) {
            customMessages[data[i].product] = {
                max: "В наличии только " +
                    data[i].stocklevel + " шт."
            }
        }

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg",
            messages: customMessages
        });

        $('input').change(function(e) {
            $('form').validate().element($(e.target));
        })
    })
</script>
...
```

В этом примере информативность диагностических сообщений повышается за счет включения в них информации, содержащейся в свойствах `stocklevel` элементов данных.

Задание текста сообщений при поэлементной проверке

Когда правила применяются к отдельным элементам, можно передавать им объект `messages`, содержащий требуемые тексты диагностических сообщений. Соответствующий пример приведен в листинге 13.19.

Листинг 13.19. Определение диагностических сообщений для правил, применяемых при поэлементной проверке

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
```

```
            stocklevel: "0", price: 1.50},
        { name: "Примулы", product: "primula",
            stocklevel: "1", price: 3.12},
        { name: "Подснежники", product: "snowdrop",
            stocklevel: "15", price: 0.99},
    ];

    var templResult = $('#flowerTmpl').tmpl(data);
    templResult.slice(0, 3).appendTo('#row1');
    templResult.slice(3).appendTo("#row2");

    $('form').validate({
        highlight: function(element, errorClass) {
            $(element).add($(element).parent())
            .addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).add($(element).parent())
            .removeClass("invalidElem");
        },
        errorElement: "div",
        errorClass: "errorMsg"
    });

    $('input').each(function(index, elem) {
        $(elem).rules("add", {
            min: 10,
            max: 20,
            messages: {
                max: "В наличии только " +
                data[index].stocklevel + " шт."
            }
        })
    }).change(function(e) {
        $('form').validate().element($(e.target));
    })
})
</script>
...
```

Здесь при определении текста сообщений также используется свойство `stocklevel`. Для простоты я предположил, что элементы `input` располагаются в том же порядке, что и элементы данных. Результат представлен на рис. 13.11.

Создание пользовательской проверки

Если встроенных проверок оказывается недостаточно, можно создать нестандартную проверку. Данный процесс сравнительно прост, а это означает, что вам не составит большого труда тесно увязать новый вид проверки со структурой своего веб-приложения. Соответствующий пример приведен в листинге 13.20.

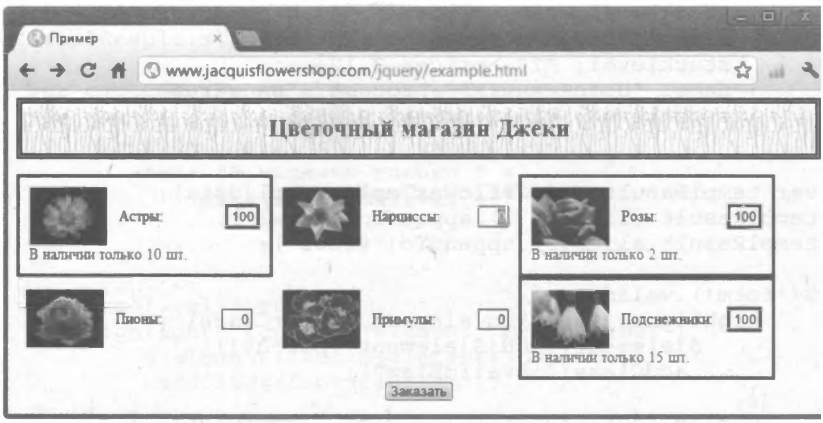


Рис. 13.11. Определение сообщений, получаемых из объекта данных

Листинг 13.20. Создание пользовательской проверки

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $.validator
```

```
            .addMethod("stock", function(value, elem, args) {
            return Number(value) <= Number(args);
        }, "Такого количества нет в наличии");

        $('input').each(function(index, elem) {
            $(elem).rules("add", {
                min: 0,
                stock: data[index].stocklevel
            })
        }).change(function(e) {
            $('form').validate().element($(e.target));
        })
    })
</script>
...
```

Пользовательская проверка определяется с помощью метода `addMethod()`, который вызывается для свойства `validator` функции `$()`. В качестве аргументов методу передаются имя, присваиваемое данному виду проверки, функция, которая используется для выполнения проверки, и сообщение, которое должно отображаться, если проверяемое условие не выполняется. В данном примере мы определили проверку под названием `stock`.

Определение функции, выполняющей проверку

Функция, выполняющая проверку, принимает в качестве аргументов значение, введенное пользователем, объект `HTMLElement`, представляющий элемент формы, и любые аргументы, которые были указаны при применении данной проверки к элементу, как показано ниже.

```
...
$(elem).rules("add", {
min: 0,
stock: data[index].stocklevel
})
...
```

При применении этого правила в качестве аргумента проверки было указано значение свойства `stocklevel`. Это значение передается в пользовательскую функцию проверки.

```
function(value, elem, args) {
    return Number(value) <= Number(args);
}
```

На корректность проверяемого значения указывает возвращаемый данной функцией результат. Если значение корректно, функция возвращает `true`. Значение (`value`) и аргументы (`args`) представляются в виде строк, и поэтому потребовалось их приведение к типу `Number`, чтобы они сравнивались в JavaScript как числа. В данном случае значение считается правильным, если оно не превышает значения аргумента.

Определение пользовательского сообщения

Сообщение, которое должно отображаться в случае ошибки, можно определить двумя способами. Первый из них — задать его в виде строки, как в предыдущем примере. Второй способ предполагает использование функции, что позволяет создавать сообщения, более соответствующие контексту. Пример приведен в листинге 13.21.

Листинг 13.21. Создание пользовательского сообщения с помощью функции

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = [
            { name: "Астры", product: "astor",
              stocklevel: "10", price: 2.99},
            { name: "Нарциссы", product: "daffodil",
              stocklevel: "12", price: 1.99},
            { name: "Розы", product: "rose",
              stocklevel: "2", price: 4.99},
            { name: "Пионы", product: "peony",
              stocklevel: "0", price: 1.50},
            { name: "Примулы", product: "primula",
              stocklevel: "1", price: 3.12},
            { name: "Подснежники", product: "snowdrop",
              stocklevel: "15", price: 0.99},
        ];

        var templResult = $('#flowerTmpl').tmpl(data);
        templResult.slice(0, 3).appendTo('#row1');
        templResult.slice(3).appendTo("#row2");

        $('form').validate({
            highlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .addClass("invalidElem");
            },
            unhighlight: function(element, errorClass) {
                $(element).add($(element).parent())
                .removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $.validator
         .addMethod("stock", function(value, elem, args) {
            return Number(value) <= Number(args);
        }, function(args) {
            return "В наличии только " + args + " шт."
        });

        $('input').each(function(index, elem) {
            $(elem).rules("add", {
                min: 0,
                stock: data[index].stocklevel
            })
        }).change(function(e) {
            $('form').validate().element($(e.target));
        })
    })
</script>
...
```

Аргументом этой функции служит аргумент, который вы предоставляете при применении правила. В данном примере таковым является значение свойства `stocklevel`. Результат представлен на рис. 13.12.

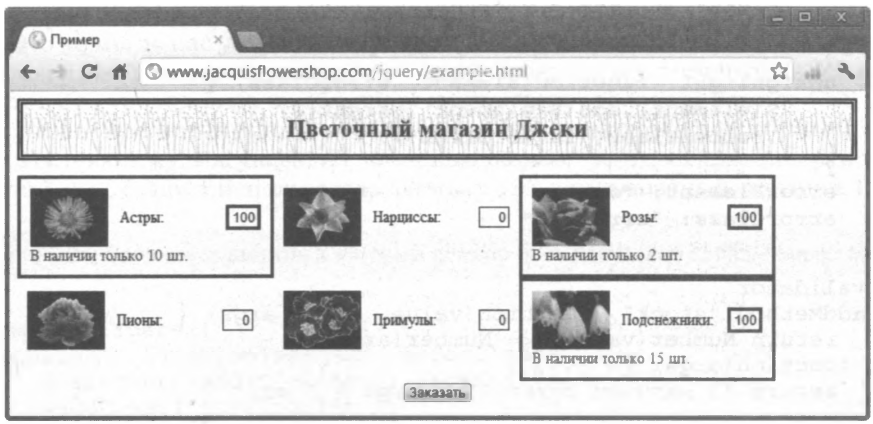


Рис. 13.12. Определение сообщений об ошибках для пользовательских проверок с помощью функции

Форматирование выводимых сообщений об ошибках

По моему мнению, одной из наилучших возможностей, предоставляемых подключаемым модулем Validation, является широкое разнообразие конфигурационных параметров, позволяющих управлять форматом сообщений об ошибках при их отображении для пользователя. Возможности настройки вывода сообщений об ошибках, которые мы использовали до сих пор, выделены в листинге 13.22 полужирным шрифтом.

Листинг 13.22. Настройка форматирования сообщений об ошибках в ходе проверки введенных данных

```
...
$(document).ready(function() {

    var data = [
        { name: "Астры", product: "astor",
          stocklevel: "10", price: 2.99},
        { name: "Нарциссы", product: "daffodil",
          stocklevel: "12", price: 1.99},
        { name: "Розы", product: "rose",
          stocklevel: "2", price: 4.99},
        { name: "Пионы", product: "peony",
          stocklevel: "0", price: 1.50},
        { name: "Примулы", product: "primula",
          stocklevel: "1", price: 3.12},
        { name: "Подснежники", product: "snowdrop",
          stocklevel: "15", price: 0.99},
    ];

    var templResult = $('#flowerTmpl').tmpl(data);
    templResult.slice(0, 3).appendTo('#row1');
```

```
    templResult.slice(3).appendTo("#row2");

    $('form').validate({
        highlight: function(element, errorClass) {
            $(element).add($(element).parent())
            .addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).add($(element).parent())
            .removeClass("invalidElem");
        },
        errorElement: "div",
        errorClass: "errorMsg"
    });

    $.validator
     .addMethod("stock", function(value, elem, args) {
        return Number(value) <= Number(args);
    }, function(args) {
        return "В наличии только " + args + " шт."
    });

    $('input').each(function(index, elem) {
        $(elem).rules("add", {
            min: 0,
            stock: data[index].stocklevel
        })
    }).change(function(e) {
        $('form').validate().element($(e.target));
    })
})
</script>
...
```

Здесь используются четыре различные конфигурационные опции, однако они тесно связаны между собой. В следующих разделах эти опции описаны по отдельности.

Задание класса для некорректных элементов

С помощью параметра `errorClass` можно задать класс, который будет связываться с некорректными элементами. Этот класс применяется к сообщениям об ошибках при их добавлении в документ. В наших примерах для этой цели используется класс `errorMessage`, которому в элементе `style` присвоен соответствующий стиль, как показано в листинге 13.23.

Листинг 13.23. Элемент `style`, используемый в образце документа

```
...
<style type="text/css">
    div.errorMsg {color: red}
    .invalidElem {border: medium solid red}
</style>
...
```

Этот стиль устанавливает для указанного класса свойство `color` таким образом, чтобы текст сообщения был красного цвета.

Задание элемента errorElement

Сообщения об ошибках вставляются в документ в качестве сестринского элемента, непосредственно следующего за тем элементом формы, который содержит некорректное значение. По умолчанию текст сообщения содержится в элементе label. Это не могло нас устроить в приводимых примерах, поскольку внешняя таблица стилей уже содержит селектор, который выбирает все элементы label внутри элементов div уровня ряда в табличной компоновке страницы в стиле CSS и применяет стиль, препятствующий отображению текста подходящим образом. Чтобы обойти эту проблему, мы указали с помощью параметра errorElement, что вместо элемента label следует использовать элемент div, как показано в листинге 13.24.

Листинг 13.24. Задание элемента, в который должно быть помещено сообщение об ошибке

```
...
$('form').validate({
    highlight: function(element, errorClass) {
        $(element).add($(element).parent())
        .addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).add($(element).parent())
        .removeClass("invalidElem");
    },
    errorElement: "div",
    errorClass: "errorMsg",
});
...
```

Задание визуального выделения некорректных элементов

Параметры highlight и unhighlight позволяют указать функции, которые должны использоваться для визуального выделения элементов, содержащих неправильные значения. Аргументами этих функций служат объект HTMLElement, представляющий некорректный элемент, и класс, который был задан с помощью параметра errorClass. Как показано в листинге 13.25, внутри указанных функций данный класс нами не используется, тогда как с помощью объекта HTMLElement мы создаем набор jQuery, после чего переходим к родительскому элементу и присваиваем ему класс invalidElem.

Листинг 13.25. Управление визуальным выделением элемента

```
...
$('form').validate({
    highlight: function(element, errorClass) {
        $(element).add($(element).parent())
        .addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).add($(element).parent())
        .removeClass("invalidElem");
    },
    errorElement: "div",
    errorClass: "errorMsg",
```

```
});
...
```

Функция, определяемая конфигурационным параметром `unhighlight`, вызывается тогда, когда пользователь устраняет ошибку и в соответствующем поле содержится корректное значение. Мы используем эту возможность для удаления класса, который ранее был добавлен другой функцией. Класс `invalidElem` соответствует селектору в элементе `style`, включенном в документ, как показано в листинге 13.26.

Листинг 13.26. Стиль визуального выделения элемента

```
...
<style type="text/css">
    div.errorMsg {color: red}
    .invalidElem {border: medium solid red}
</style>
...
```

В рассматриваемых функциях можно осуществлять любые манипуляции выбранными элементами. В данном случае рамка добавлялась к родительскому элементу исключительно для того, чтобы продемонстрировать возможности свободного обращения с DOM, но с равным успехом можно воздействовать как на сам элемент, содержащий ошибочное значение, так и вообще на любую другую часть документа, если это необходимо.

Использование отчета о проверке

Вместо того чтобы добавлять отдельные сообщения рядом с каждым из элементов, содержащих неправильные значения, можно предоставить пользователю единый список всех обнаруженных ошибок в форме. Такая возможность может быть полезной, если структура или компоновка страницы не обеспечивает необходимой гибкости для размещения дополнительных элементов. Пример создания отчета об ошибках приведен в листинге 13.27.

Листинг 13.27. Использование отчета об ошибках

```
...
<script type="text/javascript">
$(document).ready(function() {

    var data = [
        { name: "Астры", product: "astor",
          stocklevel: "10", price: 2.99},
        { name: "Нарциссы", product: "daffodil",
          stocklevel: "12", price: 1.99},
        { name: "Розы", product: "rose",
          stocklevel: "2", price: 4.99},
        { name: "Пионы", product: "peony",
          stocklevel: "0", price: 1.50},
        { name: "Примулы", product: "primula",
          stocklevel: "1", price: 3.12},
        { name: "Подснежники", product: "snowdrop",
          stocklevel: "15", price: 0.99},
    ];

    var plurals = {
```

```
        astor: "астр",
        daffodil: "нарциссов",
        rose: "роз",
        peony: "пионов",
        primula: "примул",
        snowdrop: "подснежников"
            }
    var templResult = $('#flowerTmpl').tmpl(data);
    templResult.slice(0, 3).appendTo('#row1');
    templResult.slice(3).appendTo("#row2");

    $('<div id=errorSummary>
        Пожалуйста, исправьте следующие ошибки:</div>')
        .append('<ul id="errorsList"></ul>')
        .hide().insertAfter('h1');

    $('form').validate({
        highlight: function(element, errorClass) {
            $(element).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).removeClass("invalidElem");
        },
        errorContainer: "#errorSummary",
        errorLabelContainer: "#errorList",
        wrapper: 'li',
        errorElement: "div"
    });

    $.validator
     .addMethod("stock", function(value, elem, args) {
        return Number(value) <= Number(args.data.stocklevel);
    }, function(args) {
        return "Вы запросили " + $(args.element).val() + " " +
            plurals[args.data.product] +
            ", но их в наличии только "
            + args.data.stocklevel + " шт.";
    });

    $('input').each(function(index, elem) {
        $(elem).rules("add", {
            min: 0,
            stock: {
                index: index,
                data: data[index],
                element: elem
            }
        })
    }).change(function(e) {
        $('form').validate().element($(e.target));
    })

})
</script>
...
```

В этом примере я поступлю несколько иначе и сначала покажу вам результат, а затем объясню, как он был получен. На рис. 13.13 показан отчет об ошибках в окне браузера.

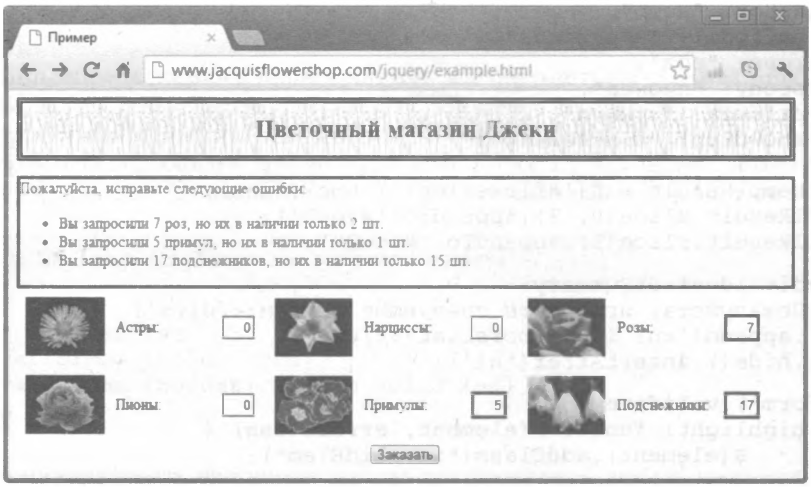


Рис. 13.13. Использование отчета о проверке корректности данных формы

Подготовка сообщений об ошибках

Первая проблема, которую нам предстоит решить при использовании отчета об ошибках, заключается в том, что контекст, доступный при расположении сообщения рядом с элементом формы, в данном случае теряется. Чтобы сообщения по-прежнему были содержательными, придется проделать дополнительную работу. Сначала определяется объект, содержащий названия цветов в родительном падеже множественного числа.

```
var plurals = {
    astor: "Астр",
    daffodil: "Нарциссов",
    rose: "Роз",
    peony: "Пионов",
    primula: "Примул",
    snowdrop: "Подснежников"
        }
```

Эти значения используются для генерации сообщений, относящихся к отдельным видам цветов, с помощью функции, определяемой при создании пользовательской проверки, как показано ниже.

```
$.validator
    .addMethod("stock", function(value, elem, args) {
    return Number(value) <= Number(args.data.stocklevel);
}, function(args) {
    return "Вы запросили " + $(args.element).val() + " " +
        plurals[args.data.product] +
        " ,но их в наличии только "
        + args.data.stocklevel + " шт.";
});
```

Связь между этими двумя стадиями обеспечивается объектом аргументов, который указывается при назначении пользовательской проверки элементам формы. Встроенные проверки имеют простые аргументы, но при выполнении пользовательской проверки можно создавать сколь угодно сложные объекты и передавать с их помощью данные любой природы.

```
$('input').each(function(index, elem) {
    $(elem).rules("add", {
        min: 0,
        stock: {
            index: index,
            data: data[index],
            element: elem
        }
    })
}).change(function(e) {
    $('form').validate().element($(e.target));
});
```

Поскольку в функциях, генерирующих сообщения, есть доступ не ко всем нужным объектам, они передаются через аргумент. В данном случае мы передаем функции следующие данные: индекс элемента, массив `data` и собственно элемент. Все эти данные включаются в сообщение, выводимое для пользователя.

Создание отчета об ошибках

Вы должны самостоятельно создать элемент, который будет содержать отчет, и добавить его в документ. В данном случае в документ добавляется элемент `div`, содержащий элемент `ul`. Наша цель — создать неупорядоченный список сообщений о каждой обнаруженной ошибке.

```
$('<div id=errorSummary>
    Пожалуйста, исправьте следующие ошибки:</div>')
    .append('<ul id="errorsList"></ul>')
    .hide().insertAfter('h1');
```

В элемент `div` включен дополнительный текст. Он будет отображаться над списком ошибок. В приведенной выше инструкции выделен вызов метода `hide()`. Решение о видимости данного элемента принимаете вы сами, и этот фактор не является критическим. Данный элемент может постоянно отображаться, но я считаю, что гораздо лучше отображать его только в том случае, когда необходимо показать ошибки, подлежащие устранению.

Теперь, когда мы проанализировали отдельные фрагменты кода, рассмотрим параметры для конфигурирования отчета.

```
$('form').validate({
    highlight: function(element, errorClass) {
        $(element).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).removeClass("invalidElem");
    },
    errorContainer: "#errorSummary",
    errorLabelContainer: "#errorList",
    wrapper: 'li',
    errorElement: "div"
});
```

Здесь контекст функций `highlight()` и `unhighlight()` сужен таким образом, чтобы выделялись лишь элементы `input`.

Параметр `errorContainer` задает селектор, который становится видимым только при обнаружении ошибок, которые нужно отобразить. В данном случае это элемент с идентификатором `errorSummary` (элемент `div`).

Параметр `errorLabelContainer` задает элемент, в который будут помещаться отдельные сообщения об ошибках. В данном случае таким элементом является элемент `ul`, поскольку мы хотим, чтобы сообщения отображались в виде списка.

Параметр `wrapper` задает элемент, который будет содержать все сообщения об ошибках. Это имеет смысл делать лишь в том случае, когда сообщения выводятся в виде списка. Наконец, элемент `errorElement` задает элемент, в котором будет содержаться текст сообщения. По умолчанию таковым является элемент `label`, но ради упрощения форматирования мы выбрали для этого элемент `div`. Результат воздействия данных параметров на вид суммарного отчета об ошибках представлен на рис. 13.13.

Модуль проверки автоматически удаляет сообщения, причины появления которых устранены, и как только все ошибки будут исправлены, пользователь сможет отправить форму. На рис. 13.14 показан отчет после устранения двух ошибок из тех трех, которые отображены на предыдущем рисунке.

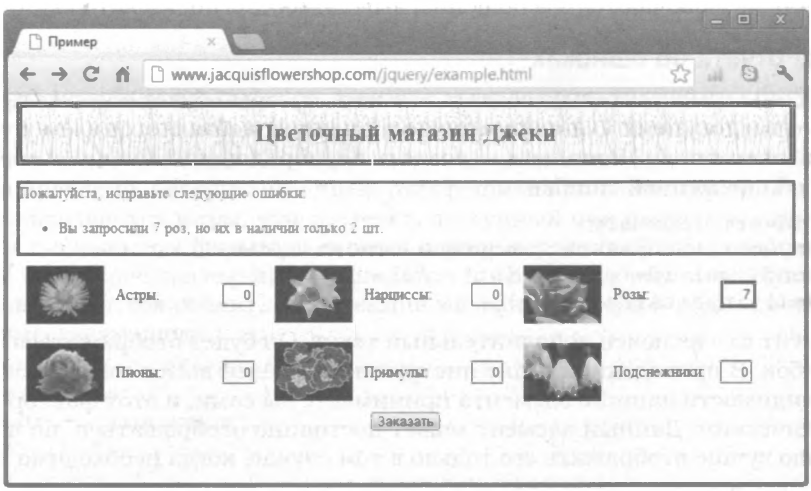


Рис. 13.14. Отчет, отображающий меньшее количество ошибок

Решение относительно того, какой вариант использовать (встроенные сообщения или отдельный сводный отчет), каждый принимает самостоятельно, и, как правило, на него оказывает влияние структура документа. Хорошая новость состоит в том, что подключаемый модуль проверки правильности данных предлагает достаточно гибкие возможности, и обычно не составляет большого труда определить и применить правила проверки, в полной мере удовлетворяющие вашим потребностям.

Резюме

В этой главе вы познакомились со средствами поддержки форм, предоставляемыми библиотекой jQuery. Глава началась с повторения методов обработки событий формы и объяснения роли наиболее важных из них в жизненном цикле HTML-формы. Большая часть главы была посвящена модулю Validation, который обеспечивает гибкую и расширяемую поддержку проверки значений, вводимых пользователем в полях формы, а также предоставляет средства, упрощающие разрешение всевозможных проблем, связанных с корректностью данных, прежде чем они будут отправлены на сервер.

ГЛАВА 14

Использование Ajax (часть I)

Ajax — это сокращение от "Asynchronous JavaScript and XML" (асинхронный Java Script и XML), но в наши дни этот термин употребляется как самостоятельный, без приписывания ему какого-либо смысла в результате расшифровки названия. Ajax позволяет отправлять на сервер асинхронные запросы. В двух словах это означает, что обмен данными между браузером и сервером выполняется в фоновом режиме, и поэтому никоим образом не мешает взаимодействию пользователя с содержимым HTML-документа. Чаще всего Ajax используется для отправки данных формы на сервер. Преимуществом такого подхода является то, что отображение ответной информации, получаемой от сервера, не требует полной перезагрузки веб-страницы, и данные документа могут беспрепятственно отображаться с помощью стандартных функций jQuery.

Средства поддержки Ajax, которые используются в этой главе, встроены в основную библиотеку jQuery, хотя в конце главы я все же приведу описание одного полезного подключаемого модуля. Наличие этой поддержки в jQuery значительно упрощает использование программного интерфейса Ajax, предусмотренного в браузерах.

В этой главе описываются так называемые *прямые* и *вспомогательные* методы Ajax. Они представляют собой упрощенные методы, обеспечивающие сравнительно простое и быстрое использование возможностей Ajax. В главе 15 описан низкоуровневый программный интерфейс jQuery Ajax API, на основе которого эти методы реализованы. Однако, как вскоре будет показано, низкоуровневый API вовсе не такой уж низкоуровневый, и им, как правило, пользуются в тех случаях, когда возможностей прямых и вспомогательных методов оказывается недостаточно. Перечень тем, рассматриваемых в данной главе, приведен в табл. 14.1.

Таблица 14.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Выполнение асинхронных HTTP-запросов GET	Используйте метод `get()`	1-3
Обработка данных, полученных посредством Ajax-запроса	Передайте функцию методу `get()`	4
Выполнение Ajax-запросов в ответ на действия пользователя	Вызовите метод `get()` в обработчике события	5
Запрос на получение данных в формате JSON от сервера	Используйте метод `get()` и получите объект в аргументе функции	6, 7
Отправка данных на сервер в виде части HTTP-запроса GET	Передайте объект JavaScript методу `get()` в качестве аргумента	8

Окончание табл. 14.1

Задача	Решение	Листинг
Выполнение асинхронных HTTP-запросов POST	Используйте метод `post()`	9, 10
Отправка не связанных с формой данных в POST-запросе	Передайте объект JavaScript методу `post()` в качестве аргумента	11
Замена типа данных, указанного сервером в ответе на Ajax-запрос	Передайте ожидаемый тип методу `get()` или `post()` в качестве аргумента	12, 13
Избежание самой распространенной ловушки Ajax	Не обращайтесь с Ajax-запросами так, словно они являются асинхронными	14
Использование вспомогательных методов для выполнения GET-запросов на получение специфических типов данных	Используйте методы `load()`, `getScript()` и `getJSON`	15–18
Разрешение использования Ajax для элементов формы	Используйте подключаемый модуль Ajax Forms	19

Использование прямых методов Ajax

Обычно средства Ajax ассоциируются с отправкой данных формы, однако их реальная сфера применения гораздо шире. Изучение методов Ajax начнем с выполнения простейших задач. Рассмотрим способы получения данных с веб-сервера без привлечения форм.

В библиотеке jQuery определен ряд так называемых *прямых методов*, которые в действительности представляют собой оболочки для вызовов функций ядра Ajax и позволяют быстро и просто решать типичные задачи Ajax. В следующих разделах вы познакомитесь с прямыми методами, предназначенными для получения данных с веб-сервера посредством HTTP-запросов GET.

Кратко об асинхронных задачах

Если вы новичок в Ajax, позвольте вкратце рассказать вам о том, что такое асинхронные запросы. Это важно знать, поскольку асинхронные запросы занимают в Ajax центральное место, а буква *A* в Ajax происходит от слова *asynchronous*. Как программист большую часть времени вы тратите на написание синхронного кода. Вы определяете блок кода, решающий некоторую задачу, и остальная часть программы ждет, пока этот блок не выполнится. Задача завершается выполнением последнего оператора блока. На то время, пока выполняется блок, браузер лишает пользователя возможности взаимодействовать с содержимым веб-страницы.

Если же задача выполняется асинхронно, то вы сообщаете браузеру, что намерены выполнить определенную работу в фоновом режиме. Выражение "в фоновом режиме" звучит довольно туманно, но для браузера оно означает следующее: "Выполни это так, чтобы пользователь в течение всего времени мог взаимодействовать с документом, и сообщи мне, когда все будет сделано". В случае Ajax вы приказываете браузеру связаться с веб-сервером и сообщить вам, когда запрос будет выполнен. Управление этой связью осуществляется с помощью *функций обратного вызова* (callback functions). Вы предоставляете браузеру одну или несколько функций, которые должны быть вызваны сразу же по завершении выполнения задачи. Должна быть предусмотрена функция, которая обработает успешный запрос, а кроме того, могут существовать функции, выполняющиеся в случае других исходов, например при возникновении ошибок.

Преимущество асинхронных запросов состоит в том, что они позволяют создавать функционально насыщенные HTML-документы, которые могут непрерывно обновляться на основании полученных от сервера ответов, не прерывая взаимодействия пользователя с приложением и не заставляя его дожидаться окончания полной загрузки документа.

Недостатком такого подхода является то, что он требует тщательного продумывания кода. Нельзя заранее сказать, когда именно будет выполнен асинхронный запрос, и вы не имеете права делать какие-либо предположения относительно возможного исхода запроса. Кроме того, использование функций обратного вызова приводит к созданию еще более сложного кода, который только и ждет, чтобы "наказать" программиста, неосторожно сделавшего какие-либо предположения относительно возможного исхода запроса или времени его выполнения.

Выполнение GET-запросов Ajax

Прежде всего, Ajax используется для того, чтобы выполнить HTTP-запрос GET с целью загрузки HTML-фрагмента, который можно добавить в документ. Образец документа, с которым мы будем работать, приведен в листинге 14.1.

Листинг 14.1. Образец документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery.validate.js"type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            // сюда будет помещаться код
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2"class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Этот код (файл `example.html`) аналогичен тому, который использовался в предыдущих примерах, но в нем отсутствуют элементы, описывающие отдельные виды цветочной продукции, а также элементы данных и шаблоны, с помощью которых они генерируются. Вместо этого создан отдельный файл под названием `flowers`.

html, который, как и файл example.html, включен в архив примеров, доступный на сайте книги (см. главу 1). Содержимое файла flowers.html приведено в листинге 14.2[1].

Листинг 14.2. Содержимое файла flowers.html

```
...
<div>
    <img src="astor.png"/><label for="astor">Астры:</label>
    <input name="astor" value="0" required />
</div>
<div>
    <img src="daffodil.png"/>
    <label for="daffodil">Нарциссы:</label>
    <input name="daffodil" value="0" required />
</div>
<div>
    <img src="rose.png"/><label for="rose">Розы:</label>
    <input name="rose" value="0" required />
</div>
<div>
    <img src="peony.png"/><label for="peony">Пионы:</label>
    <input name="peony" value="0" required />
</div>
<div>
    <img src="primula.png"/><label for="primula">Примулы:</label>
    <input name="primula" value="0" required />
</div>
<div>
    <img src="snowdrop.png"/>
    <label for="snowdrop">Подснежники:</label>
    <input name="snowdrop" value="0" required />
</div>
...
```

Это те же элементы, которые использовались в предыдущих главах, но они не распределены по рядам и из элементов div удален атрибут class. Эти изменения внесены исключительно для того, чтобы продемонстрировать, как осуществляется обработка элементов, загружаемых в документ. Никаких технических предпосылок, которые заставляли бы поступать именно так, нет. Обратите внимание на то, что это не полный HTML-документ, а всего лишь фрагмент.

Для включения этого фрагмента в основной HTML-документ мы можем воспользоваться поддержкой Ajax, предоставляемой jQuery. Возможно, вас удивляет использование такого подхода, однако мы используем его, поскольку он имитирует распространенную ситуацию, когда сложный документ или веб-приложение получают путем "сшивки" отдельных фрагментов содержимого, создаваемого различными системами. Для простоты в этом примере используется только один сервер, но нетрудно себе представить, что информация о предлагаемой продукции может поступать из нескольких разных источников. В последующих примерах я использую сервер Node.js для демонстрации того, как организовать работу с несколькими серверами. А пока что давайте познакомимся с базовыми возможностями Ajax, предоставляемыми библиотекой jQuery, и используем их для работы с файлом flowers.html. Пример того, как это можно сделать, приведен в листинге 14.3.

---

[1] Чтобы избежать проблем с отображением кириллицы, сохраните этот файл в кодировке UTF-8. — *Примеч. ред.*

Листинг 14.3. Использование средств Ajax jQuery для работы с HTML-фрагментом

```
...
<script type="text/javascript">
$(document).ready(function() {
    $.get("flowers.html",
        function(data) {
            var elems = $(data).filter('div').addClass("dcell");
            elems.slice(0, 3).appendTo('#row1');
            elems.slice(3).appendTo("#row2");
    });
});
</script>
...
```

Здесь используется метод `get()`, которому передаются два аргумента. Первый из них — это URL-адрес, указывающий на документ, который мы хотим загрузить. В данном случае используется адрес `flowers.html`, который будет интерпретироваться как URL, заданный относительно URL-адреса, использующегося для загрузки основного документа.

Второй аргумент — функция, которая будет вызываться в случае успешного выполнения запроса. Как уже отмечалось выше, в Ajax интенсивно используются функции обратного вызова, поскольку запросы выполняются в асинхронном режиме.

Когда вы будете загружать документ, содержащий этот сценарий, файл `flowers.html` загрузится с сервера, браузер выполнит его синтаксический анализ, и полученные элементы добавятся в документ. Конечный результат представлен на рис. 14.1.

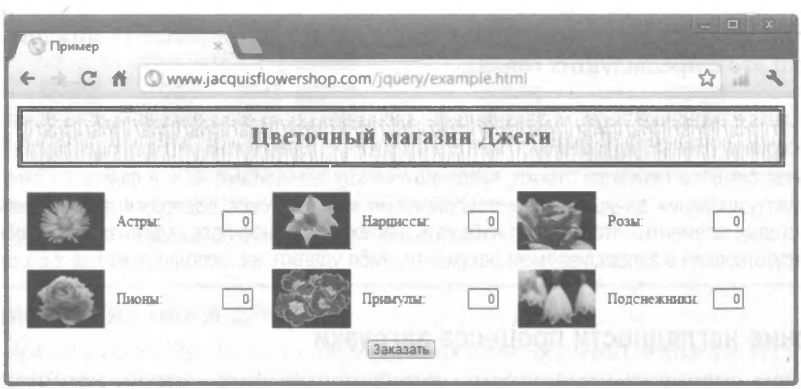


Рис. 14.1. Результат использования Ajax

Надеюсь, и вы получили тот же результат, с которым уже неоднократно сталкивались при использовании встроенных элементов и данных, однако то, как именно мы добились его сейчас, заслуживает подробного рассмотрения. Поэтому давайте углубимся в детали.

Совет. В этом сценарии метод `get()` используется для загрузки HTML-страницы, однако точно так же с сервера могут быть загружены данные любой природы.

Обработка ответных данных сервера

Функция, выполняющаяся в случае успешного завершения запроса, принимает в качестве аргумента данные, отправленные сервером в ответ на запрос. В этом примере мы получаем содержимое файла `flowers.html`, представляющее собой HTML-фрагмент. Чтобы превратить этот фрагмент в объект, с которым можно работать средствами jQuery, мы передаем его функции `$()`, которая выполнит синтаксический анализ фрагмента и сгенерирует дерево объектов `HTMLElement`, как показано в листинге 14.4.

Листинг 14.4. Обработка данных, полученных от сервера

```
...
<script type="text/javascript">
$(document).ready(function() {
    $.get("flowers.html",
        function(data) {
            var elems = $(data).filter('div').addClass("dcell");
            elems.slice(0, 3).appendTo('#row1');
            elems.slice(3).appendTo("#row2");
    });
});
</script>
...
```

Как уже отмечалось, из элементов `div` были намеренно удалены атрибуты `class`. Теперь, как видите, мы восстанавливаем их с помощью стандартного метода `addClass()`. Передав данные функции `$()`, мы получаем от нее объект `jQuery`, с которым далее можем работать как с любым другим объектом. Мы добавляем элементы в документ с помощью методов `slice()` и `appendTo()` аналогично тому, как делали это в предыдущих главах.

Совет. Обратите внимание на то, что для выбора элементов `div`, сгенерированных на основе полученных от сервера данных, используется метод `filter()`. Дело в том, что в процессе синтаксического анализа символы перевода строки, введенные между элементами `div` в файле `flowers.html` для структурирования данных, jQuery воспринимает как текстовое содержимое и вставляет вместо них текстовые элементы. Чтобы этого избежать, необходимо либо проследить за тем, чтобы эти символы отсутствовали в запрашиваемом документе, либо удалить их, используя метод `filter()`.

Повышение наглядности процесса загрузки

В нашем сценарии инструкции, запускающие Ajax-запрос, начинают выполняться, когда срабатывает событие `ready` (см. главу 9). Это мешает увидеть различия между использованием Ajax и встроенных данных. Для повышения наглядности процесса загрузки добавим в документ кнопку и сделаем так, чтобы Ajax-запрос выполнялся лишь после того, как на этой кнопке будет выполнен щелчок. Вносимые изменения отражены в листинге 14.5.

Листинг 14.5. Выполнение Ajax-запроса после щелчка на кнопке

```
...
<script type="text/javascript">
$(document).ready(function() {
    $('<button>Ajax</button>').appendTo('#buttonDiv')
```

```
    .click(function(e) {
        $.get("flowers.html",
          function(data) {
              var elems = $(data).filter('div').addClass("dcell");
              elems.slice(0, 3).appendTo('#row1');
              elems.slice(3).appendTo("#row2");
        });
        e.preventDefault();
    });
});
</script>
...
```

Теперь документ `flowers.html` не будет загружаться до тех пор, пока на кнопке Ajax не будет выполнен щелчок, причем каждый последующий щелчок также будет приводить к добавлению в документ дополнительных элементов, как показано на рис. 14.2. Обратите внимание на вызов метода `preventDefault()` для объекта `Event`, который передается в обработчик событий. Это делается для того, чтобы отменить выполнение браузером действий, предусмотренных по умолчанию. Поскольку элемент `button` содержится внутри элемента `form`, действием по умолчанию является отправка формы на сервер.

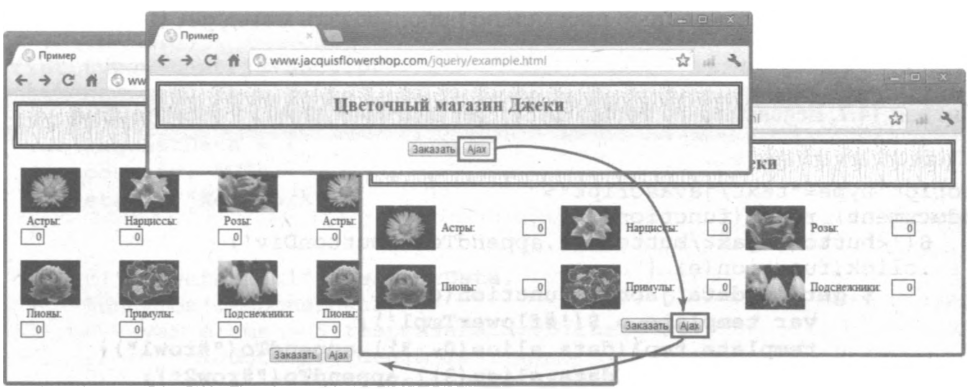


Рис. 14.2. Использование Ajax после щелчка на кнопке

Получение других типов данных

Возможности метода `get()` не ограничиваются работой только с HTML. Он позволяет получать с сервера данные любой природы. Особый интерес для нас представляет формат JSON, предоставляющий удобные возможности для обработки данных средствами jQuery. В свое время, когда технология Ajax еще только начинала широко внедряться, предпочтительным форматом данных считался XML, и даже буква *X* в аббревиатуре "Ajax" обязана своим происхождением этому формату. Я не собираюсь подробно обсуждать XML и лишь замечу, что этот язык страдает избыточностью синтаксиса, трудно читается и требует относительно больших затрат времени и ресурсов для генерации и обработки данных.

В последние годы формат XML был заметно потеснен форматом JSON (JavaScript Object Notation), отличающимся простотой и исключительной приспособленностью для работы с JavaScript-кодом (о чем говорит уже само его название). Специально для этого примера я создал файл `mydata.json` и сохранил его вместе с файлом

example.html на веб-сервере. Содержимое файла mydata.json представлено в листинге 14.6[2].

Листинг 14.6. Содержимое файла mydata.json

```
[{"name":"Астры","product":"astor","stocklevel":"10",
  "price":"2.99"},
 {"name":"Нарциссы","product":"daffodil","stocklevel":"12",
  "price":"1.99"},
 {"name":"Розы","product":"rose","stocklevel":"2",
  "price":"4.99"},
{"name":"Пионы","product":"peony","stocklevel":"0",
  "price":"1.50"},
 {"name":"Примулы","product":"primula","stocklevel":"1",
  "price":"3.12"},
 {"name":"Подснежники","product":"snowdrop","stocklevel":"15",
  "price":"0.99"}]
```

В этом файле содержатся данные для различных видов цветочной продукции, и несложно заметить, что форма представления данных JSON совпадает с формой представления данных, встраиваемых в JavaScript-код. Это и есть одна из причин, по которым формат JSON вытеснил XML в веб-приложениях. Для загрузки и обработки этих данных с помощью Ajax можно использовать метод get(), как показано в листинге 14.7.

Листинг 14.7. Использование метода get() для получения данных в формате JSON

```
...
<script type="text/javascript">
$(document).ready(function() {
    $('<button>Ajax</button>').appendTo('#buttonDiv')
    .click(function(e) {
        $.get("mydata.json", function(data) {
            var template = $('#flowerTmpl');
            template.tmpl(data.slice(0, 3)).appendTo("#row1");
            template.tmpl(data.slice(3)).appendTo("#row2");
        });
        e.preventDefault();
    });
});
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" data-price="${price}"
            data-stock="${stocklevel}" value="0" required />
    </div>
</script>
...
```

В этом примере файл с данными JSON запрашивается с сервера после щелчка на кнопке. Полученные данные передаются функции так, как если бы это был

[2] Чтобы избежать проблем с отображением кириллицы, сохраните этот файл в кодировке UTF-8. — *Примеч. ред.*

обычный HTML-фрагмент. Для обработки данных и генерации из них HTML-элементов используется подключаемый модуль шаблонов (см. главу 12), а для вставки элементов в документ — методы `slice()` и `appendTo()`. Обратите внимание на то, что мы не предпринимаем никаких действий для того, чтобы преобразовать строку JSON в объект JavaScript, — за нас это автоматически делает jQuery.

Совет. Некоторые веб-серверы (в том числе и сервер Microsoft IIS 7.5, который я использовал в примерах) не возвращают содержимое браузеру, если не могут распознать расширение имени файла или формат данных. Чтобы этот пример работал с IIS, мне пришлось установить новое соответствие между расширением имени файла (`.json`) и MIME-типом данных в формате JSON (`application/json`). Пока я этого не сделал, веб-сервер IIS отвечал на мой запрос файла `mydata.json` сообщением с кодом `404` ("Not Found").

Передача данных GET-запросам

Данные могут пересылаться на сервер в составе GET-запросов, для отправки которых можно использовать методы `get()`, `load()`, `getScript()` и `getJSON()`. Соответствующий пример приведен в листинге 14.8.

Листинг 14.8. Пересылка данных в составе GET-запроса

```
...
<script type="text/javascript">
$(document).ready(function() {

    var requestData = {
        country: "US",
        state: "New York"
    }

    $.get("flowers.html", requestData,
        function(responseData) {
            var elems = $(responseData).filter('div')
                .addClass("dcell");
            elems.slice(0, 3).appendTo('#row1');
            elems.slice(3).appendTo("#row2");
    });

});
</script>
...
```

Предоставленные вами данные присоединяются к указанному URL-адресу в виде строки запроса. Это означает, что запрос имеет примерно следующий вид:

```
http://www.jacquisflowershop.com/jquery/flowers.html?
country=US&state=New+York
```

Данные, полученные в составе запроса, могут быть использованы сервером для уточнения содержимого, которое должно быть возвращено в ответ на запрос. Например, в приведенном запросе передана информация, относящаяся к одному из отделений интернет-магазина, которые разбросаны по нескольким штатам.

GET или POST: что лучше?

У вас может возникнуть соблазн использовать метод GET для отправки данных формы на сервер. Будьте внимательны! В повседневной практике старайтесь придерживаться правила, согласно которому GET-запросы следует использовать лишь для получения данных, предназначенных только для чтения. тогда как для операций, способных изменить состояние приложения, необходимо использовать метод POST.

В стандартах говорится о том, что GET-запросы предназначены для использования в тех случаях, когда взаимодействие является безопасным (т.е. используется исключительно для получения информации и не сопровождается никакими другими побочными эффектами), в то время как POST-запросы предназначены для небезопасных видов взаимодействия (в частности, таких, которые влияют на принятие решений либо изменяют значения параметров или объектов). Эти положения установлены Консорциумом WWW (World Wide Web Consortium — W3C) и доступны для ознакомления по следующему адресу:

`http://www.w3.org/Provider/Style/URI`

Таким образом, использование GET-запросов для отправки данных формы на сервер допускается, однако они не должны предназначаться для выполнения операций, изменяющих состояние приложения. Многие разработчики усвоили это правило на своем горьком опыте в 2005 году, когда была опубликована программа Google Web Accelerator. Эта программа осуществляла предварительную загрузку содержимого по всем ссылкам на каждой странице, что абсолютно допустимо в рамках протокола HTTP, ибо ожидается, что GET-запросы должны быть безопасными. К сожалению, многие разработчики игнорировали требования HTTP и включали простые ссылки, позволяющие выполнять операции наподобие **Удалить** или **Добавить в корзину**. Разумеется, это привело ко всеобщему хаосу.

Одна компания решила, что ее система управления содержимым стала объектом злонамеренных атак, поскольку все содержимое ее веб-сайта периодически удалялось. Впоследствии оказалось, что в поле зрения робота-поисковика попал URL-адрес административной страницы, и он регулярно выбирал все ссылки, запускающие удаление данных.

Выполнение POST-запросов Ajax

Теперь, когда вы уже знаете, как получать данные с сервера, можно перейти к рассмотрению вопроса о способах отправки данных, точнее — отправки данных формы на сервер. Для решения этой задачи также предусмотрен прямой метод `post()`, значительно упрощающий отправку данных формы. Прежде чем перейти к рассмотрению данного метода, вы должны настроить свой сервер, а это означает, что мы должны вновь вернуться к серверу Node.js и постараться понять, как он работает в рамках политики безопасности, которую браузеры применяют в отношении POST-запросов Ajax.

Подготовка сервера Node.js к получению данных формы

Для этого раздела главы вам понадобится серверный сценарий, который будет получать данные, отправленные браузером с использованием HTTP-метода POST, выполнять простую операцию с использованием этих данных и генерировать ответ. Сценарий Node.js для этого раздела приведен в листинге 14.9.

Листинг 14.9. Сценарий Node.js для отправки данных на сервер методом `POST`

```
var http = require('http');
var url = require('url');
var querystring = require('querystring');

http.createServer(function (req, res) {
    console.log("[200 OK] " + req.method + " to " + req.url);
```

```
    if (req.method == 'OPTIONS') {
        res.writeHead(200, "OK",
            {"Access-Control-Allow-Headers": "Content-Type",
             "Access-Control-Allow-Methods": "*",
             "Access-Control-Allow-Origin":
                 "http://www.jacquisflowershop.com"
            });
        res.end();

    } else if (req.method == 'POST') {
        var dataObj = new Object();
        var contentType = req.headers["content-type"];
        var fullBody = '';

        if (contentType) {
            if (contentType
            .indexOf("application/x-www-form-urlencoded") > -1) {
                req.on('data',
                function(chunk) { fullBody += chunk.toString();});
                req.on('end', function() {
                    var dBody = querystring.parse(fullBody);
                    writeResponse(req, res, dBody,
                        url.parse(req.url, true)
                        .query["callback"])
                });
            } else {
                req.on('data',
                function(chunk) { fullBody += chunk.toString();});
                req.on('end', function() {
                    dataObj = JSON.parse(fullBody);
                    var dprops = new Object();
                    for (var i = 0; i < dataObj.length; i++) {
                        dprops[dataObj[i]
                        .name] = dataObj[i].value;
                    }
                    writeResponse(req, res, dprops);
                });
            }
        }
    } else if (req.method == "GET") {
        var data = url.parse(req.url, true).query;
        writeResponse(req, res, data, data["callback"])
    }
    console.log("Ready on port 9999");
}).listen(9999);

function writeResponse(req, res, data, jsonp) {
    var total = 0;
    for (item in data) {
        if(item != "_" && data[item] > 0) {
            total += Number(data[item]);
        } else {
            delete data[item];
        }
    }
    data.total = total;
    jsonData = JSON.stringify(data);
    if (jsonp) {
```

```
        jsonData = jsonp + "(" + jsonData + ")";
    }

    res.writeHead(200, "OK", {
        "Content-Type": "application/json",
        "Access-Control-Allow-Origin":
            "http://www.jacquisflowershop.com"});
    res.write(jsonData);
    res.end();
}
```

Сохраните сценарий в файле `formserver.js`. Чтобы избавить себя от ввода текста вручную, возьмите его из файлов примеров, доступных на сайте книги. Введите в командной строке следующую команду.

```
node.exe formserver.js
```

Этот сценарий обрабатывает данные, отправленные браузером, и генерирует ответ в формате JSON. Вообще говоря, можно было сделать так, чтобы этот сценарий возвращал HTML-данные, но формат JSON более компактен, и во многих случаях с ним проще работать. Возвращаемый объект JSON чрезвычайно прост: он содержит общее количество единиц продукции, выбранной пользователем, и количество заказанных единиц по тем видам продукции, для которых оно указано. Таким образом, если выбрать одну астру, два нарцисса и две розы, то ответ в формате JSON, отправляемый сценарием Node.js, будет выглядеть так:

```
{"astor":"1","daffodil":"2","rose":"2","total":3}
```

Ранее формат JSON использовался нами для представления массива объектов, тогда как данный серверный сценарий возвращает одиночный объект, свойства которого соответствуют выбранным видам цветов. Свойство `total` содержит общее количество выбранных цветов. Должен признать, что подобного рода операции слишком просты для того, чтобы дать полное представление о возможностях обработки данных на сервере, но все же основной предмет нашего рассмотрения — Ajax, а не вопросы разработки серверных приложений.

Кроссдоменные запросы Ajax

Если вы заглянете в серверный сценарий (файл `formserver.js`), то увидите, что в ответ сервера браузеру вставляется следующий HTTP-заголовок:

```
Access-Control-Allow-Origin: http://www.jacquisflowershop.com/
```

По умолчанию браузеры разрешают сценариям выполнение Ajax-запросов, только если они имеют общий *источник происхождения* (origin) с документом, который их содержит[3]. Здесь под источником происхождения понимается комбинация протокола, имени хоста и номера порта, являющихся компонентами URL-адреса. Если перечисленные компоненты двух URL-адресов совпадают, то эти адреса соответствуют одному и тому же источнику. Если же они отличаются значением хотя бы одного компонента, то считается, что они относятся к разным источникам.

[3] Так называемое *правило ограничения домена* (same origin policy), согласно которому URL-адрес ответа на запрос должен принадлежать тому же домену, что и сервер, на котором находится страница, запрашивающая ответ. — *Примеч. ред.*

Совет. Такая политика направлена на уменьшение риска атак посредством *межсайтовых сценариев* (cross-site scripting — XSS[4]), когда браузер (или пользователь), введенный в заблуждение злоумышленниками, выполняет включенный в обычные страницы вредоносный сценарий. В этой книге атаки CSS не рассматриваются, однако в Википедии имеется превосходная статья на эту тему:

`http://en.wikipedia.org/wiki/Cross-site_scripting`

В табл. 14.2 приведены результаты сравнения ряда URL-адресов с URL-адресом базового образца документа:

`www.jacquisflowershop.com/jquery/example.html`

Таблица 14.2. Сравнение URL-адресов

URL-адрес	*Сравнение источников*
`http://www.jacquisflowershop.com/apps/mydoc.html`	Совпадают
`https://www.jacquisflowershop.com/apps/mydoc.html`	Различны; отличаются протоколом
`http://www.jacquisflowershop.com:81/apps/mydoc.html`	Различны; отличаются номерами портов
`http://node.jacquisflowershop.com/order`	Различны; отличаются хостами

Конфигурация моей системы включает два сервера. Один из них, `www.jacquisflowershop.com`, обрабатывает статическое содержимое, тогда как на втором, `node.jacquisflowershop.com`, выполняется сценарий Node.js. Как следует из данных таблицы, любой документ, хранящийся на первом сервере, и второй сервер относятся к разным источникам. Запросы, направляемые из одного источника к другому, отличному от него, называются *кроссдоменные запросы* (cross-domain requests).

Проблемы, связанные с политикой ограничения доменов, обусловлены тем, что она устанавливает полный запрет на использование кроссдоменных запросов — их просто не должно быть, и точка! Это заставило разработчиков искать всевозможные обходные пути, позволяющие "обмануть" браузер и вынудить его выполнять запросы, идущие вразрез с указанной политикой. К счастью, существуют вполне легальные возможности выполнения кроссдоменных запросов, устанавливаемые спецификацией *CORS* (Cross-Origin Resource Sharing). Ниже приводится лишь краткое описание CORBS. Для получения более полных сведений по этому вопросу обратитесь к самой спецификации, которая доступна по адресу `http://www.w3.org/TR/cors`.

Совет. Разработка спецификации CORBS была завершена сравнительно недавно. Она поддерживается текущими поколениями браузеров, тогда как старые браузеры могут игнорировать кроссдоменные запросы. Традиционный подход, основанный на использовании схемы JSONP, которую мы еще будем обсуждать, подходит для всех типов браузеров.

В сценарии Node.js (файл `formserver.js`) источник `www.jacquisflowershop.com` указан в качестве доверенного в заголовке `Access-Control-Allow-Origin` непосредственно в коде, но ничто не мешает получать значение этого заголовка в запросе, что позволяет организовывать более сложные процессы принятия решений. Кроме того, в этом заголовке можно использовать звездочку (*), что равносильно разрешению выполнять кроссдоменные запросы из любых источников. Этим

[4] Обычно, чтобы не возникало путаницы с каскадными таблицами стилей, при ссылках на межсайтовый скриптинг используют аббревиатуру XSS. — *Примеч. ред.*

удобно пользоваться на этапе тестирования, но если вы намерены оставить такую возможность в производственном приложении, то предварительно тщательно продумайте, не приведет ли это к созданию уязвимостей в системе безопасности.

Использование метода POST для отправки данных формы

Итак, теперь, когда вы уже имеете подготовленный сервер и знаете, что такое CORS, можем приступить к использованию метода `post()` для оправки данных формы на сервер, как показано в листинге 14.10.

Листинг 14.10. Отправка данных с помощью метода `post()`

```
...
<script type="text/javascript">
$(document).ready(function() {

    $('button').get(0).disabled = true;

    $.getJSON("mydata.json", function(data) {
        var template = $('#flowerTmpl');
        template.tmpl(data.slice(0, 3)).appendTo("#row1");
        template.tmpl(data.slice(3)).appendTo("#row2");
        $('button').get(0).disabled = false;
    });

    $('button').click(function(e) {
        var formData = $('form').serialize();

        $.post("http://node.jacquisflowershop.com:9999/
            order", formData,
            function(data) {
                processServerResponse(data);
            })

        e.preventDefault();
    })

    function processServerResponse(data) {
        var inputElems = $('div.dcell').hide();

        for (var prop in data) {
            var filtered = inputElems.has('input[name=' +
                prop + ']').appendTo("#row1").show();
        }

        $('#buttonDiv, #totalDiv').remove();
        $('#totalTmpl').tmpl(data).appendTo('body');
    }
});
</script>
<script id="totalTmpl" type="text/x-jquery-tmpl">
    <div id="totalDiv" style="clear: both; padding: 5px">
        <div style="text-align: center">Total Items:
        <span id=total>${total}</span></div>
        <div id="buttonDiv"><button type="submit">Заказать
            </button></div>
    </div>
</script>
```

```
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" data-price="${price}"
            data-stock="${stocklevel}" value="0" required />
    </div>
</script>
...
```

Этот пример кажется немного сложнее, чем он есть на самом деле. Мы начинаем с того, что используем метод `getJSON()` для получения файла `mydata.json`, содержащего описание цветочной продукции, после чего генерируем элементы с помощью шаблона данных и добавляем их в документ. В результате мы оказываемся на уже хорошо вам знакомой исходной позиции, которую, надеюсь, вы уже успели полюбить (рис. 14.3).

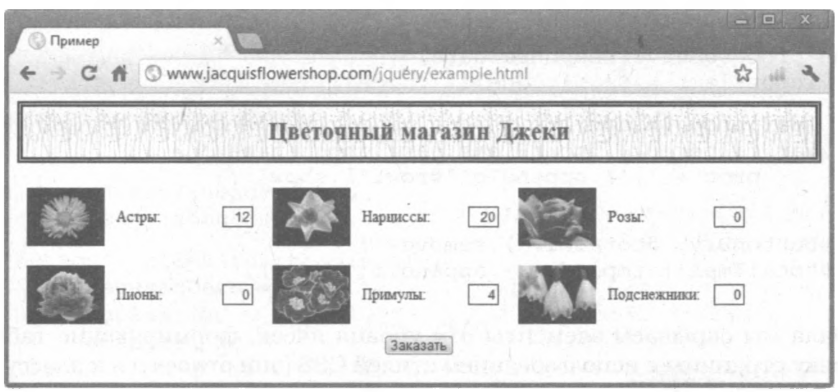


Рис. 14.3. Исходная позиция перед отправкой данных на сервер

Вы видите на этом рисунке, что в некоторых текстовых полях содержатся значения, формирующие заказ: 12 астр, 20 нарциссов и 4 примулы. Для регистрации функции, которая будет вызываться после щелчка на кнопке, используется метод `click()`.

```
$('button').click(function(e) {
    var formData = $('form').serialize();

    $.post("http://node.jacquisflowershop.com:9999/
        order", formData,
        function(data) {
            processServerResponse(data);
        })

    e.preventDefault();
})
```

Первое, что делает данная функция, — вызывает метод `serialize()` для элемента `form`. Это весьма полезный метод, который последовательно обрабатывает все элементы формы и создает строку, закодированную для передачи на сервер в качестве фрагмента URL-адреса.

Для введенных мною значений метод `serialize()` генерирует следующую строку:

```
astor=12&daffodil=20&rose=0&peony=0&primula=4&snowdrop=0
```

Совет. Обратите внимание на вызов метода `preventDefault()` для объекта `Event`, который передается в обработчик событий. Этот вызов необходим для того, чтобы предотвратить отправку формы браузером обычным путем, т.е. путем отправки данных и полной перезагрузки документа для получения ответа.

Я использую здесь метод `serialize()`, поскольку метод `post()` отправляет данные в закодированном для передачи в составе URL формате (хотя можно поступить иначе и использовать глобальный обработчик событий `ajaxSetup()`, о котором пойдет речь в главе 15). Получив объект `data`, созданный на основе значений элементов `input`, мы вызываем метод `post()`, чтобы инициировать Ajax-запрос.

В качестве аргументов метод `post()` получает URL-адрес, по которому должны быть отправлены данные (он должен совпадать с URL-адресом, указанным в атрибуте `action` элемента `form`), подлежащие отправке, и функцию, которая должна быть вызвана в случае успешного выполнения запроса. В этом примере получаемый от сервера ответ передается функции `processServerResponse()`, которая определена следующим образом.

```
function processServerResponse(data) {
    var inputElems = $('div.dcell').hide();

    for (var prop in data) {
        var filtered = inputElems.has('input[name=' +
            prop + ']').appendTo("#row1").show();
    }

    $('#buttonDiv, #totalDiv').remove();
    $('#totalTmpl').tmpl(data).appendTo('body');
}
```

Сначала мы скрываем элементы `div` уровня ячеек, формирующие табличную компоновку страницы с использованием стилей CSS (они относятся к классу `dcell`), а затем отображаем те из них, которые соответствуют свойствам объекта JSON, полученного от сервера. Кроме того, мы используем шаблон данных для генерации разметки, отображающей общее количество выбранных единиц продукции. Это все можно было бы сделать с помощью клиента, но наша цель — научиться обрабатывать данные, возвращаемые POST-запросом Ajax. Результат показан на рис. 14.4.

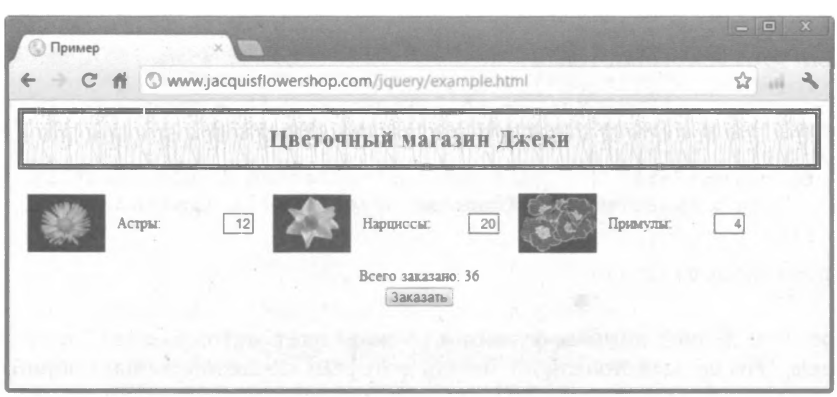


Рис. 14.4. Результат обработки данных, возвращаемых сервером в ответ на POST-запрос Ajax

Теперь вы сами могли убедиться в том, насколько легко отправить данные формы на сервер (и, конечно же, насколько просто обработать ответ, особенно если он возвращается в формате JSON).

Совет. Если получить ответ, представленный на рисунке, вам не удается, то это может быть связано с тем, что в заголовке CORS в сценарии Node.js не установлено корректное имя домена.

Отправка других данных с использованием метода POST

Несмотря на то что обычно метод `post()` применяется для отправки данных формы, разрешается отправлять с его помощью практически любые необходимые данные. Для этого требуется лишь создать объект, содержащий данные, вызвать метод `serialize()` для их подходящего форматирования и передать их методу `post()`. Эта методика может пригодиться, если вы получаете данные от пользователя, минуя форму, или хотите включить в POST-запрос лишь часть данных, введенных в элементах формы. Использование метода `post()` таким способом продемонстрировано в листинге 14.11.

Листинг 14.11. Использование метода `post()` для отправки на сервер данных, не являющихся элементами формы

```
...
<script type="text/javascript">
$(document).ready(function() {

    $('button').click(function(e) {
        var requestData = {
            apples: 2,
            oranges: 10
        };

        $.post("http://node.jacquisflowershop.com/
            order", requestData,
            function(responseData) {
                alert(JSON.stringify(responseData));
            })
        e.preventDefault();
    })
});
</script>
...
```

В этом сценарии явным образом создается объект и определяются его свойства. Этот объект передается методу `post()`, а для отображения полученного от сервера ответа используется метод `alert()`. (В действительности серверу совершенно безразлично, данные какого типа он получает от браузера. Он просто попытается их сложить и получить итоговую сумму.) На рис. 14.5 показано диалоговое окно с выведенным результатом.

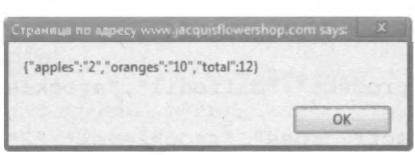


Рис. 14.5. Ответ сервера на получение данных, не являющихся элементами формы

Совет. Полученный от сервера ответ в формате JSON автоматически преобразуется в объект JavaScript. Для обратного преобразования этого объекта в строку, которую можно отобразить в диалоговом окне, в сценарии используется метод `JSON.stringify()`.

Указание ожидаемого типа данных

При использовании методов `get()` и `post()` библиотеке jQuery приходится определять тип данных, получаемых от сервера в ответ на запрос. Данными может быть все что угодно, начиная от HTML-кода и заканчивая файлами JavaScript. Для определения типа данных библиотека jQuery использует содержащуюся в ответе информацию, и в частности — заголовок `Content-Type`. Как правило, этого вполне достаточно, но иногда jQuery приходится оказывать небольшую помощь. Обычно необходимость в этом возникает из-за указания сервером неверного MIME-типа в ответе.

Можно изменить информацию, поставляемую сервером, и сообщить jQuery, какой тип данных ожидается, передавая методам `get()` и `post()` дополнительную информацию. Аргумент может принимать одно из следующих значений:

- `xml`;
- `json`;
- `jsonp`;
- `script`;
- `html`;
- `text`.

В листинге 14.12 показано, как задать ожидаемый тип данных для метода `get()`.

Листинг 14.12. Указание ожидаемого типа данных

```
...
<script type="text/javascript">
$(document).ready(function() {

    $.get("mydata.json",
        function(responseData) {
            console.log(JSON.stringify(responseData));
    }, "json");

});
</script>
...
```

В случае использования коротких форм вызова Ajax, обеспечиваемых прямыми методами, тип данных задается последним аргументом. Здесь мы сообщаем jQuery, что рассчитываем получить данные в формате JSON. Сведения о типе содержимого, предоставляемые сервером, игнорируются, и jQuery будет обрабатывать ответ как объект JSON. В примере ответ сервера выводится на консоль и выглядит так.

```
[{"name":"Астры","product":"astor","stocklevel":"10",
  "price":"2.99"},
 {"name":"Нарциссы","product":"daffodil","stocklevel":"12",
  "price":"1.99"},
 {"name":"Розы","product":"rose","stocklevel":"2",
  "price":"4.99"},
 {"name":"Пионы","product":"peony","stocklevel":"0",
```

```
  "price":"1.50"},
 {"name":"Примулы","product":"primula","stocklevel":"1",
  "price":"3.12"},
 {"name":"Подснежники","product":"snowdrop","stocklevel":"15",
  "price":"0.99"}]
```

Это содержимое совпадает с тем, которое было помещено в файл `mydata.json`, что, собственно говоря, и ожидалось. Единственное, что требуется от вас при указании типа данных, — это абсолютная точность. Несоответствие типа реальных данных указанному типу может стать источником проблем, как показано в листинге 14.13.

Листинг 14.13. Указание неверного типа данных

```
...
<script type="text/javascript">
$(document).ready(function() {

    $.get("flowers.html",
        function(responseData) {
            console.log(JSON.stringify(responseData));
    }, "json");

});
</script>
...
```

В этом примере мы запрашиваем файл, содержащий HTML-код, но сообщаем jQuery, что содержимое следует обрабатывать как объект JSON. Проблема в том, что при обработке объекта JSON библиотека jQuery автоматически создает объект JavaScript, чего не может сделать с HTML-кодом. В результате Ajax-запрос закончится выводом следующего сообщения об ошибке:

```
SyntaxError: Unexpected token <
```

Совет. О способах обнаружения ошибок Ajax говорится в главе 15.

Коварная ловушка при работе с Ajax

Прежде чем мы продолжим обсуждение, хочу продемонстрировать последствия самой распространенной ошибки, которую допускают веб-программисты при работе с Ajax, обрабатывая асинхронный запрос так, как если бы он был синхронным. Пример возникновения этой проблемы представлен в листинге 14.14.

Листинг 14.14. Распространенная ошибка при работе с Ajax

```
...
<script type="text/javascript">
$(document).ready(function() {
    $('<button>Ajax</button>').appendTo('#buttonDiv')
        .click(function(e) {
        e.preventDefault();

        var elems;

        $.get("flowers.html", function(data) {
```

```
            elems = $(data).filter("div").addClass("dcell");
        });

        elems.slice(0, 3).appendTo('#row1');
        elems.slice(3).appendTo("#row2");

    });
});
</script>
...
```

В этом сценарии определяется переменная `elems`, которая используется в функции обратного вызова Ajax для сохранения результата выполнения запроса к серверу. Полученные с сервера элементы добавляются в документ с помощью методов `slice()` и `appendTo()`. Если вы выполните этот пример, то увидите, что ни один из элементов не будет добавлен в документ, и вместо этого на консоли отобразится сообщение, конкретный текст которого зависит от типа браузера. Ниже приведено сообщение, отображаемое на консоли браузера Google Chrome.

```
Uncaught TypeError: Cannot call method 'slice' of undefined
```

Проблема заключается в том, что инструкции, содержащиеся внутри элемента `script`, выполняются не в том порядке, в котором они записаны. При написании этого кода предполагался следующий порядок выполнения инструкций.

1. Определяется переменная `elems`.
2. Получаемые с сервера данные присваиваются переменной `elems`.
3. Элементы извлекаются из переменной `elems` и добавляются в документ.

В действительности происходит следующее.

1. Определяется переменная `elems`.
2. Запускается асинхронный запрос к серверу.
3. Элементы извлекаются из переменной `elems` и добавляются в документ.

При этом в какой-то промежуточный момент времени вскоре после отправки браузером запроса происходит следующее.

1. В браузер поступают данные от сервера.
2. Данные обрабатываются и присваиваются переменной `elems`.

Причина появления сообщения об ошибке — вызов метода `slice()` для переменной в тот момент, когда она еще не содержит никаких элементов. Хуже всего то, что иногда этот код может работать правильно. Объясняется это тем, что Ajax-запрос выполняется настолько быстро, что к тому времени, когда начинается обработка переменной, в ней уже содержатся ожидаемые данные (обычно это наблюдается в тех случаях, когда данные кешируются браузером или когда между отправкой Ajax-запроса и попыткой обработки данных выполняются какие-либо сложные операции). Так что теперь, если ваш код будет вести себя подобным образом, вы уже знаете, что может быть причиной его необычного поведения.

Использование вспомогательных методов для работы с конкретными типами данных

Библиотека jQuery предоставляет три вспомогательных метода, которые делают работу с некоторыми типами данных более удобной. Описанию и демонстрации использования этих методов посвящены несколько следующих разделов.

Получение HTML-фрагментов

Метод `load()` предназначен для получения *только* HTML-данных, что позволяет совместить запрос HTML-фрагмента, обработку ответа от сервера для создания набора элементов и вставку этих элементов в документ в одном действии. Пример использования метода `load()` представлен в листинге 14.15.

Листинг 14.15. Использование прямого метода `load()`

```
...
<script type="text/javascript">
$(document).ready(function() {
    $('<button>Ajax</button>').appendTo('#buttonDiv')
    .click(function(e) {
        $('#row1').load("flowers.html");
        e.preventDefault();
    });
});
</script>
...
```

В этом сценарии мы вызываем метод `load ()` для элемента, в который хотим вставить новые элементы, и передаем ему URL-адрес в качестве аргумента. Если запрос завершается успешно, а полученный от сервера ответ содержит действительный HTML-фрагмент, элементы вставляются в указанное место в документе, как показано на рис. 14.6.

Вы видите, что все элементы из файла `flowers.html` добавлены в документ, как мы и хотели, но поскольку у них отсутствует атрибут `class`, то они не укладываются в табличную компоновку страницы, используемую в основном документе. Поэтому метод `load()` наиболее полезен в тех случаях, когда все элементы могут быть вставлены в одно место в документе без какой-либо дополнительной обработки.

Получение и выполнение сценариев

Метод `getScript()` загружает файл JavaScript, а затем выполняет содержащиеся в нем инструкции. Чтобы продемонстрировать работу этого метода, я создал файл `myscript.js` и сохранил его вместе с файлом `example.html` на своем веб-сервере. Содержимое этого файла представлено в листинге 14.16[5].

[5] Чтобы избежать проблем с отображением кириллицы, сохраните этот файл в кодировке UTF-8. — *Примеч. ред.*

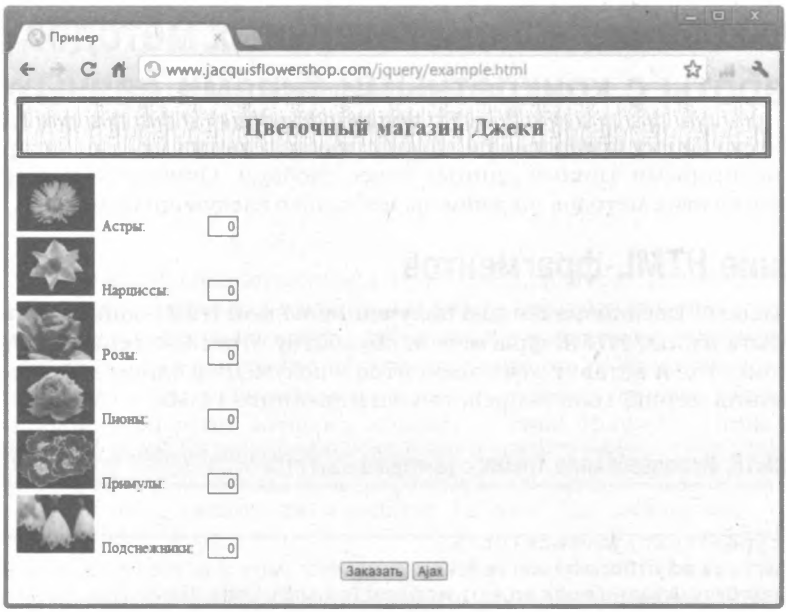


Рис. 14.6. Добавление элементов в документ с помощью метода `load()`

Листинг 14.16. Содержимое файла `myscript.js`

```
var flowers = [
    ["Astor", "Daffodil", "Rose"],
    ["Peony", "Primula", "Snowdrop"],
    ["Carnation", "Lily", "Orchid"]
]
var flowersR = [
    ["Астры", "Нарциссы", "Розы"],
    ["Пионы", "Примулы", "Подснежники"],
    ["Гвоздики", "Лилии", "Орхидеи"]
]

$('<div id=row3 class=drow/>').appendTo('div.dtable');

var fTemplate = $('<div class=dcell><img/><label/><input/>
                   </div>');

for (var row = 0; row < flowers.length; row++) {
    var fNames = flowers[row];
     var fNamesR = flowersR[row];

    for (var i = 0; i < fNames.length; i++) {
        fTemplate.clone().appendTo("#row" + (row + 1)).children()
            .filter('img').attr('src', fNames[i] + ".png").end()
            .filter('label').attr('for', fNames[i])
            .text(fNamesR[i]).end()
            .filter('input').attr({name: fNames[i], value: 0})
    }
}
```

Эти инструкции генерируют три ряда элементов, описывающих цветы. Мы обошлись здесь без определения шаблонов и использовали циклы для генерации элементов (хотя, вообще говоря, следовало бы воспользоваться шаблонами данных, описанными в главе 12).

Самое важное, что необходимо знать при работе со сценариями, — между инициализацией Ajax-запроса и выполнением инструкций сценария состояние документа может измениться. В листинге 14.17 приведен сценарий из основного документа, в котором по-прежнему используется метод `getScript()`, но при этом, еще до завершения Ajax-запроса, модифицируется дерево DOM.

Листинг 14.17. Отправка запроса и одновременное выполнение сценария при использовании метода `getScript()`

```
...
<script type="text/javascript">
$(document).ready(function() {
    $('<button>Ajax</button>').appendTo('#buttonDiv')
    .click(function(e) {
        $.getScript("myscript.js");
        $('#row2').remove();

        e.preventDefault();
    });
});
</script>
...
```

Здесь мы вызываем метод `getScript()` для основной функции `$()` и передаем ему в качестве аргумента URL-адрес файла, который хотим использовать. Если сервер способен предоставить указанный файл и этот файл содержит действительный JavaScript-код, то последний будет выполнен.

Совет. Метод `getScript()` можно использовать для загрузки любых сценариев, но особенно полезно использовать его для загрузки и выполнения таких сценариев, как сценарии для отслеживания статистики посещения сайтов или определения географического местоположения клиента, которые не связаны с поддержкой основной функциональности веб-приложения. Пользователя не особенно заботит, в состоянии ли мы точно определять его местоположение для ведения статистики, тогда как длительное ожидание загрузки и выполнения сценария будет действовать ему на нервы. Используя метод `getScript()`, можно быстро получать запрашиваемую информацию, не доставляя пользователям неудобств, вызванных необходимостью ожидания ответа. Уточню свою мысль. Я вовсе не предлагаю вам использовать этот метод для выполнения каких-либо действий скрытно от пользователя и говорю лишь о том, что следует отодвигать на второй план загрузку и выполнение вполне законной функциональности, если она представляет для пользователей меньшую ценность, чем затрачиваемое на ее ожидание время.

В данном примере после запуска Ajax-запроса с помощью метода `getScript()` из документа удаляется элемент `row2`, для чего используется метод `remove()`. Данный элемент используется в файле `myscript.js` для вставки новых элементов. Эти элементы отбрасываются незаметным для пользователя образом, поскольку в документе селектору `#row2` ничто не соответствует. Итоговый результат представлен на рис. 14.7. Отнеситесь к этому примеру как к одному из образцов надежного дизайна для ситуаций, в которых документ подвергается изменениям. Во всяком случае запомните, что при написании внешних сценариев JavaScript не стоит делать слишком много допущений о состоянии документа.

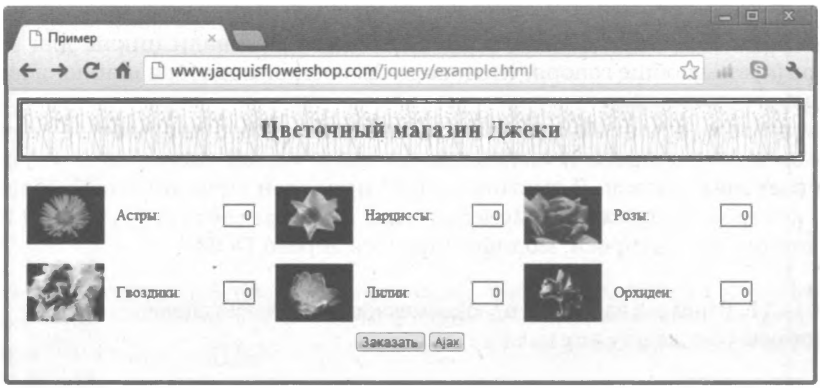


Рис. 14.7. Результат внесения изменений в документ во время выполнения запроса

Получение данных в формате JSON

Для загрузки данных JSON с сервера предназначен метод `getJSON()`. Возможно, это наименее полезный из всех трех вспомогательных методов, поскольку он не делает с данными ничего сверх того, что делает базовый метод `get()`. Пример использования метода `getJSON()` приведен в листинге 14.18.

Листинг 14.18. Использование метода `getJSON()`

```
...
<script type="text/javascript">
$(document).ready(function() {
    $('<button>Ajax</button>').appendTo('#buttonDiv')
    .click(function(e) {
        $.getJSON("mydata.json", function(data) {
            var template = $('#flowerTmpl');
            template.tmpl(data.slice(0, 3)).appendTo("#row1");
            template.tmpl(data.slice(3)).appendTo("#row2");
        });
        e.preventDefault();
    });
});
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" data-price="${price}"
            data-stock="${stocklevel}"
            value="0" required />
    </div>
</script>
...
```

В этом примере данные JSON запрашиваются после щелчка на кнопке Ajax. Данные, загруженные с сервера, передаются функции во многом аналогично тому, как это делается при работе с методом `get()`, с которым вы познакомились ранее.

Для обработки данных и генерации элементов используется подключаемый модуль шаблонов (см. главу 12), а для вставки элементов в документ — методы `slice()` и `appendTo()`. Обратите внимание на то, что в качестве аргумента функции передается объект JavaScript. Для преобразования данных из формата JSON в объект вам ничего не нужно делать, поскольку все заботы об этом берет на себя jQuery.

Работа с JSONP

JSONP — это альтернатива CORS, предназначенная для обхода политики ограничения домена в Ajax-запросах, т.е. для выполнения кроссдоменной загрузки. Она базируется на том факте, что браузеры позволяют загружать JavaScript-код с любого сервера, и именно так работает элемент `script`, когда вы указываете атрибут `src`. Сначала в документе определяется функция, которая будет обрабатывать данные.

```
...
function processJSONP(data) {
    //...здесь обрабатываются данные...
}
...
```

Далее выполняется запрос к серверу. Строка запроса включает данные формы и свойство `callback`, в котором указывается имя функции, которую вы перед этим определили.

```
http://node.jacquisflowershop.com/order?
callback=processJSONP&astor=1
```

Сервер, который должен понимать, как работает JSONP, генерирует данные JSON, как обычно, а затем создает инструкцию JavaScript, которая вызывает созданную вами функцию и передает ей данные в качестве аргумента.

```
processJSONP({"astor":"1","daffodil":"2","rose":"2","total":5})
```

Кроме того, сервер задает для содержимого ответа тип `text/javascript`, тем самым сообщая браузеру, что отправленные данные представляют собой инструкции JavaScript, подлежащие выполнению. Результатом этого является вызов определенного ранее метода, включенного в состав данных, отправленных сервером. Благодаря этому удается обойти проблемы кроссдоменных запросов без использования CORS.

Предупреждение. Правило ограничения доменов введено не зря. Избегайте бездумного использования JSONP. Это может привести к серьезным проблемам безопасности.

В библиотеке jQuery предусмотрена очень удобная поддержка JSONP. Все, что вам нужно сделать, — это вызвать метод `getJSON()` и указать URL-адрес, содержащий выражение `callback=?` в строке запроса. Библиотека jQuery автоматически создает функцию, присвоив ей случайное имя, и использует ее при связи с сервером, откуда следует, что вам вообще не нужно модифицировать свой код. Пример создания JSONP-запроса приведен в листинге 14.19.

Листинг 14.19. Создание JSONP-запроса с использованием метода `getJSON()`

```
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $('button').get(0).disabled = true;

        $.getJSON("mydata.json", function(data) {
            var template = $('#flowerTmpl');
            template.tmpl(data.slice(0, 3)).appendTo("#row1");
            template.tmpl(data.slice(3)).appendTo("#row2");
            $('button').get(0).disabled = false;
        });

        $('button').click(function(e) {
            var formData = $('form').serialize();

            $.getJSON("http://node.jacquisflowershop.com
                /order?callback=?", formData,
                processServerResponse)

            e.preventDefault();
        })

        function processServerResponse(data) {
            var inputElems = $('div.dcell').hide();
            for (var prop in data) {
                var filtered = inputElems
                .has('input[name=' + prop + ']')
                .appendTo("#row1").show();
            }
            $('#buttonDiv, #totalDiv').remove();
            $('#totalTmpl').tmpl(data).appendTo('body');
        }
    });
</script>
```

Использование подключаемого модуля Ajax Forms

До сих пор мы использовали встроенные возможности jQuery для работы с Ajax. Как было сказано ранее, одной из сильных сторон библиотеки jQuery является простота ее расширения с целью включения новой функциональности и, как следствие, большое многообразие подключаемых модулей. В завершение главы я кратко опишу один полезный подключаемый модуль для работы с формами.

Если для отправки данных формы на сервер вы намерены использовать исключительно Ajax, то, возможно, вам пригодится подключаемый модуль Ajax Forms, который доступен для загрузки по следующему адресу:

`http://www.malsup.com/jquery/form`

Этот подключаемый модуль предельно упрощает работу с формами с использованием средств Ajax, как показано в листинге 14.20.

Листинг 14.20. Использование подключаемого модуля Ajax Forms

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery.validate.js"type="text/javascript">
    </script>
    <script src="jquery.form.js"type="text/javascript">
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            $.getScript("myscript.js");

            $('form').ajaxForm(function(data) {
                console.log(JSON.stringify(data))
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2"class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

В этом примере мы добавляем в документ файл сценария `jquery.form.js` (входит в загрузочный пакет подключаемого модуля) и вызываем в элементе `script` метод `ajaxForm()` для элемента `form`. Аргументом этого метода является функция обратного вызова, и это обеспечивает возможность доступа к ответу сервера. Это простой и наглядный подход к работе с базовыми формами Ajax, характеризующийся тем, что URL-адрес для отправки формы берется из самого элемента `form`.

Данный подключаемый модуль обладает гораздо более широкими возможностями и даже включает определенную поддержку валидации форм, но если речь идет о том, чтобы брать на себя управление Ajax-запросами, то я рекомендовал бы использовать низкоуровневые средства Ajax, которые описываются в главе 15. Однако для простых ситуаций этот модуль очень удобен в использовании и хорошо спроектирован.

Резюме

В этой главе вы познакомились с прямыми и вспомогательными методами для работы с Ajax, которые поддерживаются библиотекой jQuery. Было показано, как использовать методы `get()` и `post()` для выполнения HTTP-запросов GET и POST, описаны методы, предназначенные для работы с данными JSON, и продемонстрировано, как использовать вспомогательные методы, ориентированные на работу с

конкретными типами данных. Также вы узнали о том, какую ошибку чаще всего допускают при работе с Ajax, что такое кроссдоменные запросы и как организовать работу с ними. Кроме того, был кратко описан подключаемый модуль jQuery, который упрощает использование средств Ajax совместно с формами в еще большей степени, чем прямые методы. В следующей главе описывается низкоуровневый интерфейс для работы с Ajax, хотя, как вы сами сможете убедиться, он не такой уж низкоуровневый, и его использование не представляет сложностей.

ГЛАВА 15

Использование Ajax (часть II)

В этой главе описан низкоуровневый программный интерфейс jQuery Ajax API. Казалось бы, термин *низкоуровневый* указывает на то, что вы получаете доступ к скрытым возможностям механизма запросов, но это не совсем так. Описываемые здесь методы менее удобны по сравнению с рассмотренными в главе 14, однако ценой небольших дополнительных усилий запрос можно сконфигурировать так, чтобы он в точности соответствовал вашим потребностям, чего не всегда удается добиться с помощью прямых или вспомогательных методов.

Перечень тем, рассматриваемых в данной главе, приведен в табл. 15.1.

Таблица 15.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Вызов Ajax с помощью низкоуровневого API	Используйте метод `ajax()`	1
Получение детальных сведений о запросе способом, аналогичным использованию собственного объекта `XMLHttpRequest`	Используйте объект `jqXHR`	2
Задание URL-адреса для Ajax-запроса	Используйте параметр `url()`	3
Задание HTTP-метода для запроса	Используйте параметр `type()`	4
Ответные действия при успешном запросе	Используйте метод `success()`	5
Ответные действия при неудачном запросе	Используйте метод `error()`	6
Ответные действия при завершении запросов, независимо от того, успешны ли они	Используйте метод `complete()`	7, 8
Настройка параметров запроса перед его отправкой	Используйте метод `beforeSend()`	9
Задание нескольких функций для обработки успешных, неудачных и любых завершенных запросов	Задайте массив функций для параметров `success`, `error` и `complete`	10
Задание элемента, который должен быть назначен переменной `this` в функциях, указанных для параметров `success`, `error` и `complete`	Используйте параметр `context`	11
Ответные действия на события для всех Ajax-запросов	Используйте методы глобальных событий	12
Задание того, должен ли запрос приводить к запуску событий `global`	Используйте параметр `global`	13
Установка тайм-аута для запроса	Используйте параметр `timeout`	14

Окончание табл. 15.1

Задача	Решение	Листинг
Добавление заголовков в запрос	Используйте параметр `headers`	14
Задание типа содержимого, отправляемого на сервер	Используйте заголовок `contentType`	15
Задание режима (синхронного или асинхронного), в котором должен выполняться запрос	Используйте параметр `async`	16
Игнорирование данных, которые не изменились	Используйте параметр `ifModified`	17
Ответные действия на получение кода состояния HTTP, отправленного сервером	Используйте метод `statusCode()`	18
Очистка данных ответа	Используйте параметр `dataFilter`	19
Управление преобразованием данных	Используйте метод `converters()`	20
Определение общих параметров для всех Ajax-запросов	Используйте метод `ajaxSetup()`	21
Динамическое изменение параметров отдельных запросов	Используйте метод `ajaxPrefilter()`	22

Создание простого Ajax-запроса средствами низкоуровневого API

Создавать запросы с помощью низкоуровневого API не намного сложнее, чем с помощью прямых или вспомогательных методов, которые обсуждались в главе 14. Разница состоит в том, что такой подход позволяет контролировать многие другие аспекты запроса и получать о выполняющемся запросе гораздо больше информации. Центральное место в низкоуровневом API занимает метод `ajax()`, простой пример использования которого приведен в листинге 15.1.

Листинг 15.1. Использование метода `ajax()`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery.validate.js"type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
        $.ajax("mydata.json",{
                success: function(data) {
                    var template = $('#flowerTmpl');
                    template.tmpl(data.slice(0, 3))
                        .appendTo("#row1");
                    template.tmpl(data.slice(3))
                        .appendTo("#row2");
```

```
                }
            });
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}" value="0" required />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2"class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Аргументами метода `ajax()` являются запрашиваемый URL и объект отображения данных, свойства которого определяют набор пар "ключ–значение", каждая из которых определяет некий параметр запроса[1].

Примечание. В этой главе используется тот же серверный сценарий Node.js, что и в главе 14.

Здесь передаваемый методу `ajax()` объект содержит только один параметр, `success`, задающий функцию, которая будет вызываться в случае успешного выполнения запроса. В данном примере мы запрашиваем у сервера файл `mydata.json` и используем его вместе с шаблоном данных для создания элементов и вставки их в документ, как это делалось в предыдущей главе с помощью прямых методов. По умолчанию метод `ajax()` создает HTTP-запрос GET, т.е. данный пример эквивалентен использованию методов `get()` и `getJSON()`, описанных в главе 14. О настройке POST-запросов будет говориться позже.

Для Ajax-запросов предусмотрено множество настроек. Все они будут рассмотрены в оставшихся главах наряду с некоторыми полезными методами, которые jQuery предоставляет для упрощения работы с Ajax.

---

[1] Заметьте, что функция `ajax()` вызывается без селектора, поскольку она не применяется к объекту Query. Все операции Ajax — глобальные функции, не зависящие от DOM. — *Примеч. ред.*

Объект jqXHR

Метод ajax() возвращает объект jqXHR, который можно использовать для получения подробной информации о запросе и с которым можно взаимодействовать. Объект jqXHR представляет собой оболочку объекта XMLHttpRequest, составляющую фундамент браузерной поддержки Ajax и приспособленную для работы с *отсроченными объектами* jQuery, о которых речь пойдет в главе 35.

При выполнении большинства операций Ajax объект jqXHR можно просто игнорировать, что я и рекомендую делать. Этот объект используется в тех случаях, когда необходимо получить более полную информацию об ответе сервера, чем та, которую удается получить иными способами. Кроме того, его можно использовать для настройки параметров Ajax-запроса, но это проще сделать, используя настройки, доступные для метода ajax(). Свойства и методы объекта jqXHR описаны в табл. 15.2.

Таблица 15.2. Свойства и методы объекта jqXHR

Свойство/метод	*Описание*
readyState	Возвращает индикатор хода выполнения запроса на протяжении всего его жизненного цикла, принимающий значения от 0 (запрос не отправлен) до 4 (запрос завершен)
status	Возвращает код состояния HTTP, отправленный сервером
statusText	Возвращает текстовое описание кода состояния
responseXML	Возвращает ответ в виде XML (если он является XML-документом)
responseText	Возвращает ответ в виде строки
setRequest(*имя, значение*)	Возвращает заголовок запроса (это можно сделать проще с помощью параметра headers)
getAllResponseHeaders()	Возвращает в виде строки все заголовки, содержащиеся в ответе
getResponseHeaders(*имя*)	Возвращает значение указанного заголовка ответа
abort()	Прерывает запрос

Объект jqXHR встречается в нескольких местах кода. Сначала он используется для сохранения результата, возвращаемого методом ajax(), как показано в листинге 15.2.

Листинг 15.2. Использование объекта jqXHR

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jqxhr = $.ajax("mydata.json",{
            success: function(data) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3))
                .appendTo("#row1");
                template.tmpl(data.slice(3))
                .appendTo("#row2");
            }
        });
```

```
        var timerID = setInterval(function() {
            console.log("Статус: " + jqxhr.status + " " +
                jqxhr.statusText);
            if (jqxhr.readyState == 4) {
                console.log("Запрос выполнен: " +
                    jqxhr.responseText);
                clearInterval(timerID);
            }
        }, 100);
    });
    </script>
...
```

В этом примере мы сохраняем результат, возвращаемый методом `ajax()`, а затем используем метод `setInterval()` для вывода информации о запросе каждые 100 мс. Использование результата, возвращаемого методом `ajax()`, не изменяет того факта, что запрос выполняется асинхронно, поэтому при работе с объектом `jqXHR` необходимо соблюдать меры предосторожности. Для проверки состояния запроса мы используем свойство `readyState` (завершению запроса соответствует значение 4) и выводим ответ сервера на консоль. Для данного сценария консольный вывод выглядит так (в вашем браузере он может выглядеть несколько иначе).

```
Статус: 200 OK

Запрос выполнен:
[{"name":"Астры","product":"astor","stocklevel":"10",
 "price":"2.99"},
{"name":"Нарциссы","product":"daffodil","stocklevel":"12",
 "price":"1.99"}, {"name":"Розы","product":"rose","stocklevel":"2",
 "price":"4.99"}, {"name":"Пионы","product":"peony","stocklevel":"0",
 "price":"1.50"},
{"name":"Примулы","product":"primula","stocklevel":"1",
 "price":"3.12"},
{"name":"Подснежники","product":"snowdrop","stocklevel":"15",
 "price":"0.99"}]
```

Я использую объект `jqXHR` лишь в редких случаях и не делаю этого вообще, если он представляет собой результат, возвращаемый методом `ajax()`. Библиотека jQuery автоматически запускает Ajax-запрос при вызове метода `ajax()`, и поэтому я не считаю возможность настройки параметров запроса сколько-нибудь полезной. Если я хочу работать с объектом `jqXHR` (как правило, для получения дополнительной информации об ответе сервера), то обычно делаю это через параметры обработчика событий, о которых мы поговорим далее. Они предоставляют мне информацию о состоянии запроса, что избавляет от необходимости выяснять его.

Задание URL-адреса запроса

Одним из наиболее важных доступных параметров является параметр `url`, позволяющий указать URL-адрес для запроса. Можно использовать этот параметр как альтернативу передаче URL-адреса в качестве аргумента метода `ajax()`, как показано в листинге 15.3.

Листинг 15.3. Задание URL-адреса

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "mydata.json",
            success: function(data) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3))
                .appendTo("#row1");
                template.tmpl(data.slice(3))
                .appendTo("#row2");
            }
        });
```

Создание POST-запроса

Для задания требуемого типа запроса, который необходимо выполнить, используется параметр `type`. По умолчанию выполняются GET-запросы, как в предыдущем примере. Пример использования метода `ajax()` для создания POST-запроса и отправки данных формы на сервер приведен в листинге 15.4.

Листинг 15.4. Создание POST-запроса с помощью метода `ajax()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "mydata.json",
            success: function(data) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            }
        });

        $('button').click(function(e) {
            $.ajax({
                url: $('form').attr("action"),
                data: $('form').serialize(),
                type: 'post',
                success: processServerResponse
            })
            e.preventDefault();
        })

        function processServerResponse(data) {
            var inputElems = $('div.dcell').hide();
                for (var prop in data) {
                var filtered = inputElems
```

```
                    .has('input[name=' + prop + ']')
                    .appendTo("#row1").show();
            }
            $('#buttonDiv, #totalDiv').remove();
            $('#totalTmpl').tmpl(data).appendTo('body');
        }
    });
</script>
<script id="totalTmpl" type="text/x-jquery-tmpl">
    <div id="totalDiv" style="clear: both; padding: 5px">
        <div style="text-align: center">Всего заказано:
            <span id=total>${total}</span></div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </div>
</script>
...
```

Здесь дополнительно к `type` мы использовали еще несколько параметров. Для указания цели POST-запроса используется описанный ранее параметр `url`. В данном случае `url` получает значение из цели элемента `form` документа. Пересылаемые данные указываются с помощью параметра `data`, значение которого устанавливается с помощью метода `serialize()`, описанного в главе 33.

Выход за рамки методов GET и POST

В параметре `type` можно указать любой HTTP-метод, но если вы попытаетесь использовать методы, отличные от GET и POST, то можете столкнуться с трудностями. Это происходит потому, что многие брандмауэры и серверы приложений конфигурируются так, чтобы любые другие виды запросов игнорировались. Если вы хотите использовать другие HTTP-методы, то можете выполнить POST-запрос, добавив в него заголовок `X-HTTP-Method-Override` и задав в нем метод, который хотите использовать, примерно так, как показано ниже.

```
X-HTTP-Method-Override: PUT
```

Это соглашение поддерживается многими фреймворками и является общепринятым способом создания веб-приложений, поддерживающих архитектурный стиль REST, — *RESTful-приложений*, более подробную информацию о которых можно найти по следующему адресу:

```
http://en.wikipedia.org/wiki/Representational_state_transfer
```

О том, как установить заголовок для Ajax-запроса jQuery, будет говориться далее.

Работа с событиями Ajax

Несколько параметров позволяют указывать функции для обработки событий, которые могут запускаться на протяжении жизненного цикла Ajax-запроса. Именно таким способом вы будете указывать функции обратного вызова, играющие столь важную роль в Ajax-запросах. С одной из них вы уже познакомились при рассмотрении параметра `success` в предыдущем примере. Список параметров, связанных с событиями, вместе с их краткими описаниями приведен в табл. 15.3.

Совет. Параметры, описанные в табл. 15.3, относятся к локальным событиям, т.е. используются для отдельных событий Ajax. Существуют также *глобальные события*, о которых речь пойдет далее.

Таблица 15.3. Параметры событий Ajax

Параметр	*Описание*
`beforeSend`	Задает функцию, которая будет вызываться перед запуском Ajax-запроса
`complete`	Задает функцию, которая будет вызываться при успешном или неудачном завершении Ajax-запроса
`error`	Задает функцию, которая будет вызываться при неудачном завершении запроса
`success`	Задает функцию, которая будет вызываться при успешном выполнении запроса

Обработка успешных запросов

В примере, поясняющем использование параметра `success`, при вызове функции были опущены два аргумента — сообщение, описывающее результат запроса, и объект `jqXHR`. Пример использования функции, которая принимает эти аргументы, приведен в листинге 15.5.

Листинг 15.5. Передача полного набора аргументов функции, определяемой параметром `success`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "mydata.json",
            success: function(data, status, jqxhr) {

                console.log("Статус: " + status);

                console.log("Статус jqXHR: " + jqxhr.status +
                    " " + jqxhr.statusText);
                console.log(jqxhr.getAllResponseHeaders());

                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            }
        });
    });
</script>
...
```

Аргумент `status` — это строка, описывающая исход запроса. Функция обратного вызова, которую мы задаем, используя параметр `success`, выполняется лишь для успешных запросов, и поэтому значением данного аргумента обычно является `success`. Исключением является случай, когда вы используете параметр `ifModified`, описанный далее.

Функции обратного вызова для всех событий Ajax следуют одному и тому же образцу, но наибольшую пользу этот аргумент приносит в случае ряда других событий.

Последний аргумент — это объект `jqXHR`. Вы не должны выяснять состояние запроса, прежде чем начать работу с этим объектом, поскольку знаете, что функция выполняется лишь тогда, когда запрос завершается успешно. В этом примере объект `jqXHR` используется для получения информации о состоянии запроса и заголовках,

которые сервер включил в ответ, а также для вывода этой информации на консоль. В данном случае результат имеет следующий вид (в зависимости от того, какой сервер вы используете, у вас может быть другой набор заголовков).

```
Статус: success
Статус jqXHR: 200 OK

Date: Sat, 23 Jun 2012 18:28:11 GMT
Connection: Keep-Alive
Content-Length: 480
Last-Modified: Thu, 21 Jun 2012 13:11:12 GMT
Server: Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8
mod_perl/2.0.4 Perl/v5.10.1
ETag: "20000000098df-1e0-4c2fb3ffcf909"
Content-Type: application/json
Accept-Ranges: bytes
Keep-Alive: timeout=5, max=97
```

Обработка ошибок

Параметр `error` используется для указания функции, которая должна вызываться при неудачном завершении запроса. Соответствующий пример приведен в листинге 15.6.

Листинг 15.6. Использование параметра `error`

```
...
<style type="text/css">
    .error {color: red; border: medium solid red; padding: 4px;
            margin: auto; width: 200px; text-align: center}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("NoSuchFile.json",{
            success: function(data, status, jqxhr) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            },
            error: function(jqxhr, status, errorMsg) {
                $('<div class=error/>')
                    .text("Статус: " + status + "   Ошибка:
                        " + errorMsg)
                    .insertAfter('h1');
            }
        });
    });
</script>
...
```

Здесь запрашивается отсутствующий на сервере файл `NoSuchFile.json`, и поэтому запрос заведомо не сможет быть выполнен, в результате чего будет вызвана функция, заданная с помощью параметра `error`. Аргументами этой функции являются объект `jqXHR`, а также сообщение о состоянии ошибки и сообщение об ошибке, полученное

в ответе сервера. Внутри этой функции в документ добавляется элемент `div`, отображающий значения аргументов `status` и `errorMsg`, как показано на рис. 15.1.

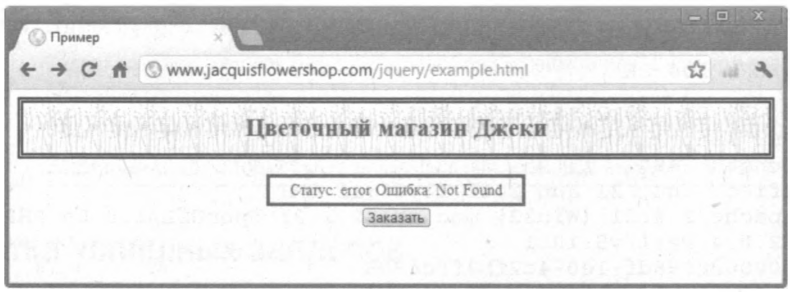


Рис. 15.1. Сообщение об ошибке

Аргумент `status` может иметь одно из значений, приведенных в табл. 15.4.

Таблица 15.4. Возможные значения `status` для параметра `error`

Параметр	*Описание*
`abort`	Запрос был отменен еще до его отправки (используется объект `jqXHR`)
`error`	Общая ошибка, о которой обычно сообщает сервер
`parsererror`	Невозможность синтаксического разбора данных, возвращенных сервером
`timeout`	Истечение допустимого времени ожидания ответа сервера

Значение аргумента `errorMsg` варьируется в зависимости от значения аргумента `status`. Если значением `status` является `error`, то аргумент `errorMsg` будет содержать текстовую часть ответа сервера. Так, в данном примере сервер отвечает сообщением `404 Not Found`, и поэтому содержимым аргумента `errorMsg` является `Not Found`.

Если значением `status` является `timeout`, то значением аргумента `errorMsg` также будет `timeout`. Можно задать длительность допустимого периода ожидания до истечения времени запроса с помощью параметра `timeout`, описанного далее.

Если значением `status` является `parsererror`, то аргумент `errorMsg` будет содержать описание проблемы. Эта ошибка встречается тогда, когда либо данные неправильно сформированы, либо сервер возвращает неправильный MIME-тип данных. Тип данных можно принудительно задать с помощью параметра `dataType`. Наконец, в случае преждевременного прекращения запроса аргументы `status` и `errorMsg` будут иметь значение `abort`.

Совет. Несмотря на то что в этом примере значения `status` и `errorMsg` отображаются в документе, в целом это мало что дает пользователю, поскольку сообщения будут понятны лишь тому, кто понимает, что происходит внутри веб-приложения, и не содержат никаких указаний относительно способов разрешения возникшей проблемы.

Обработка завершенных запросов

Параметр `complete` можно использовать для задания функции, которая будет вызываться при завершении Ajax-запроса, независимо от его успешности. Соответствующий пример приведен в листинге 15.7.

Листинг 15.7. Использование параметра `complete`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json",{
            success: function(data, status, jqxhr) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            },
            error: function(jqxhr, status, errorMsg) {
                $('<div class=error/>')
                    .text("Статус: " + status + "   Ошибка:
                        " + errorMsg)
                    .insertAfter('h1');
            },
            complete: function(jXHR, status) {
                console.log("Завершение: " + status);
                }
        });
    });
</script>
...
```

Функция, заданная параметром `complete`, вызывается после функций, заданных параметрами `success` и `error`. Для этой функции в jQuery предусмотрено намного меньше предустановленной информации, однако вы получаете гораздо более широкий ряд значений аргумента `status`, как показано в табл. 15.5.

Таблица 15.5. Возможные значения `status` для параметра `complete`

Параметр	*Описание*
`abort`	Запрос был отменен еще до его отправки (используется объект `jqXHR`)
`error`	Общая ошибка, о которой обычно сообщает сервер
`notmodified`	Запрошенное содержимое не изменялось со времени выполнения последнего запроса
`parsererror`	Невозможность синтаксического разбора данных, возвращенных сервером
`success`	Запрос был выполнен успешно
`timeout`	Истечение допустимого времени ожидания ответа сервера

У вас может возникнуть соблазн использовать параметр `complete` для задания единой функции, способной обрабатывать все возможные исходы запроса, но в этом случае вы не сможете воспользоваться преимуществами той обработки данных и ошибок, которую выполняет jQuery. Гораздо лучше использовать параметры `success` и `error` и тщательно продумать организацию аргументов общей функции, как показано в листинге 15.8.

Листинг 15.8. Использование единой функции для обработки всех возможных исходов запросов

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json",{
            success: function(data, status, jqxhr) {
                handleResponse(status, data, null, jqxhr);
            },
            error: function(jqxhr, status, errorMsg) {
                handleResponse(status, null, errorMsg, jqxhr);
            }
        });

        function handleResponse(status, data, errorMsg, jqxhr) {
            if (status == "success") {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3))
                    .appendTo("#row1");
                template.tmpl(data.slice(3))
                    .appendTo("#row2");
            } else {
                $('<div class=error/>')
                    .text("Статус: " + status + " Ошибка: " +
                        errorMsg)
                    .insertAfter('h1');
            }
        }
    });
</script>
...
```

Настройка параметров запросов перед их отправкой

Параметр `beforeEnd` позволяет задать функцию, которая будет вызываться перед отправкой запросов. Это позволяет сконфигурировать запрос в последнюю минуту, добавляя или заменяя параметры, переданные методу `ajax()` (что может быть особенно полезным, если для множества запросов используется один и тот же объект, содержащий необходимые значения параметров). Пример использования такого подхода представлен в листинге 15.9.

Листинг 15.9. Использование параметра `beforeSend`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            success: function(data, status, jqxhr) {
                handleResponse(status, data, null, jqxhr);
            },
            error: function(jqxhr, status, errorMsg) {
```

```
                handleResponse(status, null, errorMsg, jqxhr);
            },
            beforeSend: function(jqxhr, settings) {
                settings.url = "mydata.json";
            }
        });

        function handleResponse(status, data, errorMsg, jqxhr) {
            if (status == "success") {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3))
                    .appendTo("#row1");
                template.tmpl(data.slice(3))
                    .appendTo("#row2");
            } else {
                $('<div class=error/>')
                    .text("Статус: " + status + " Ошибка: " +
                        errorMsg)
                    .insertAfter('h1');
            }
        }
    });
</script>
...
```

Аргументами указанной функции являются объект `jqXHR` (который может пригодиться для настройки заголовков запроса или отмены запроса, прежде чем он будет отправлен) и объект, содержащий параметры, переданные методу `ajax()`. В данном примере URL-адрес для Ajax-запроса задается с помощью параметра `url`.

Задание нескольких обработчиков событий

В предыдущих примерах мы реагировали на наступление событий, связанных с Ajax-запросами, вызовом одной функции, но в параметрах `success`, `error`, `complete` и `beforeSend` можно задавать массив функций, каждая из которых будет выполняться при запуске соответствующего события. Простой пример этого приведен в листинге 15.10.

Листинг 15.10. Задание нескольких обработчиков событий

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json",{
            success: [processData, reportStatus]
        });

        function processData(data, status, jqxhr) {
            var template = $('#flowerTmpl');
            template.tmpl(data.slice(0, 3))
                .appendTo("#row1");
            template.tmpl(data.slice(3))
                .appendTo("#row2");
        }

        function reportStatus(data, status, jqxhr) {
           console.log("Статус: " + status + " Код результата: " +
```

```
                jqxhr.status);
        }
    });
</script>
...
```

В этом примере для параметра `success` задан массив, состоящий из двух функций, одна из которых использует данные для добавления элементов в документ, а вторая выводит информацию на консоль.

Совет. Объект `jqXHR` можно использовать для регистрации обработчиков событий в случае отсроченных объектов как часть обычной библиотеки jQuery, о чем подробно говорится в главе 35.

Настройка контекста для событий

Параметр `context` позволяет указать элемент, который будет назначен переменной `this`, когда будет вызван обработчик события. Это может быть использовано для обращения к целевым элементам в документе без необходимости их выбора в функции-обработчике. Соответствующий пример приведен в листинге 15.11.

Листинг 15.11. Использование параметра `context`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
            context: $('h1'),
            success: function(data, status, jqxhr) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            },
            complete: function(jqxhr, status) {
                var color = status == "success" ? "green" : "red";
                this.css("border", "thick solid " + color);
            }
        });
    });
</script>
...
```

Здесь параметр `context` устанавливается на объект jQuery, содержащий элементы `h1` документа. В функции, определяемой параметром `complete`, мы выделяем рамкой выбранные элементы (в данном случае — элемент, поскольку в документе есть только один элемент `h1`) путем вызова метода `css()` для объекта jQuery (на который ссылаемся через `this`). Цвет рамки определяется на основании состояния запроса.

Совет. С помощью параметра `context` можно установить в качестве контекста любой объект, и ответственность за выполнение только допустимых для этого объекта операций лежит на вас. Например, если вы задаете в качестве контекста элемент `HTMLElement`, то до того, как вызывать для него какие-либо методы jQuery, вы должны передать этот объект функции `$()`.

Использование глобальных событий Ajax

Кроме локальных событий, обрабатываемых в рамках текущего запроса, которые были описаны в предыдущей главе, в jQuery определен набор *глобальных событий*, которые можно использовать для мониторинга всех Ajax-запросов, выполняемых приложением. Список методов, доступных для работы с глобальными событиями, приведен в табл. 15.6.

Таблица 15.6. Методы jQuery, предназначенные для работы с событиями Ajax

Метод	*Описание*
`ajaxComplete(`*функция*`)`	Регистрирует функцию, которая будет вызываться всякий раз при завершении Ajax-запроса (независимо от того, был ли он успешным)
`ajaxError(`*функция*`)`	Регистрирует функцию, которая будет вызываться, если при выполнении Ajax-запроса произошла ошибка
`ajaxSend(`*функция*`)`	Регистрирует функцию, которая будет вызываться перед выполнением Ajax-запроса
`ajaxStart(`*функция*`)`	Регистрирует функцию, которая будет вызываться всякий раз при завершении Ajax-запроса (независимо от того, был ли он успешным)
`ajaxStop(`*функция*`)`	Регистрирует функцию, которая будет вызываться по завершении всех Ajax-запросов
`ajaxSuccess(`*функция*`)`	Регистрирует функцию, которая будет вызываться при успешном завершении Ajax-запроса

Как и в случае методов для обычных событий, перечисленные методы могут вызываться для любого элемента в документе. Вызываемому методу необходимо передать функцию, которая должна выполняться при наступлении события. Методы `ajaxStart()` и `ajaxStop()` не передают своим функциям никаких аргументов. Другие методы передают следующие аргументы:

- объект `Event`, содержащий описание события;
- объект `jqXHR`, содержащий описание события;
- объект, содержащий конфигурационные параметры запроса.

Совет. В документации по jQuery ошибочно утверждается, что функциям, которые передаются методам `ajaxComplete()` и `ajaxSuccess()`, предоставляются объекты `XMLHTTPRequest`, а не `jqXHR`. Это не соответствует истине: всем функциям, которые принимают аргументы, предоставляется объект `jqXHR`.

С этими методами связаны два важных момента, которые нужно хорошо запомнить. Во-первых, функции будут запускаться для событий, поступающих от *всех* Ajax-запросов, и поэтому вы всегда должны следить за тем, чтобы не делать никаких допущений, которые могут быть справедливыми только в отношении какого-либо конкретного запроса. Во-вторых, эти методы должны быть вызваны до того, как вы начнете выполнять Ajax-запросы, чтобы функции запускались, как положено. В случае вызова глобальных методов после вызова метода `ajax()` существует риск того, что Ajax-запрос закончится прежде, чем jQuery сможет нормально зарегистрировать ваши функции-обработчики. Пример использования методов для работы с глобальными событиями Ajax приведен в листинге 15.12.

Листинг 15.12. Использование методов для работы с глобальными событиями Ajax

```
...
<style type="text/css">
    .ajaxinfo {color: blue; border: medium solid blue;
        padding: 4px; margin: auto; margin-bottom:2px;
        width: 200px; text-align: center}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('<div class=ajaxinfo ><label for="globalevents">События:
         <input type="checkbox"' + 'id="globalevents"
         name="globalevents" checked></label></div>')
            .insertAfter('h1');
        $('<div id="info" class=ajaxinfo/>').text("Ready")
            .insertAfter('h1');

        $(document)
            .ajaxStart(function() {
                displayMessage("Ajax Start")
            })
            .ajaxSend(function(event, jqxhr, settings) {
                displayMessage("Ajax Send: " + settings.url)
            })
            .ajaxSuccess(function(event, jqxhr, settings) {
                displayMessage("Ajax Success: " + settings.url)
            })
            .ajaxError(function(event, jqxhr, settings, errorMsg) {
                displayMessage("Ajax Error: " + settings.url)
            })
            .ajaxComplete(function(event, jqxhr, settings) {
                displayMessage("Ajax Complete: " + settings.url)
            })
            .ajaxStop(function() {
                displayMessage("Ajax Stop")
            })

        function displayMessage(msg) {
           $('#info').queue(function() {
                $(this).fadeTo("slow", 0).queue(function() {
                        $(this).text(msg).dequeue()
                    }).fadeTo("slow", 1).dequeue();
           })
        }

        $('button').click(function(e) {
            $('#row1, #row2').children().remove();
            $.ajax("mydata.json", {
                global: $('#globalevents:checked').length > 0,
                success: function(data, status, jqxhr) {
                    var template = $('#flowerTmpl');
                    template.tmpl(data.slice(0, 3))
                        .appendTo("#row1");
                    template.tmpl(data.slice(3))
                        .appendTo("#row2");
```

```
            }
        });
        e.preventDefault();
    })
});
</script>
...
```

В этом примере регистрируются функции для всех глобальных событий Ajax. Каждая из этих функций вызывает функцию `displayMessage()` и выводит информацию о том, какое событие запущено. Поскольку Ajax-запросы могут выполняться очень быстро, я использовал очередь эффектов для замедления перехода от одного сообщения к другому, чтобы вам было легче следить за последовательностью их появления (замедление касается лишь отображаемых сообщений, но не самих Ajax-запросов). Наконец, чтобы вы могли управлять запуском последовательности событий, я добавил обработчик события `click` для кнопки. Результат представлен на рис. 15.2.

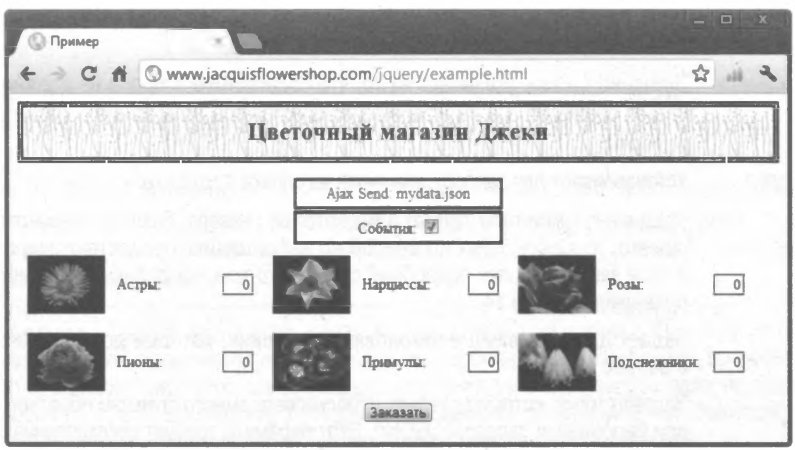


Рис. 15.2. Отображение глобальных событий Ajax

Управление глобальными событиями

Вы, наверное, заметили, что в документ был добавлен флаговый переключатель. Я использую его для установки значения параметра `global`, как показано в листинге 15.13.

Листинг 15.13. Использование параметра `global`

```
$.ajax("mydata.json", {
    global: $('#globalevents:checked').length > 0,
    success: function(data, status, jqxhr) {
        var template = $('#flowerTmpl');
        template.tmpl(data.slice(0, 3)).appendTo("#row1");
        template.tmpl(data.slice(3)).appendTo("#row2");
    }
});
```

Если параметр `global` равен `false`, то глобальные события Ajax не генерируются Ajax-запросом. В этом несложно убедиться, сделав следующее. Снимите флажок,

щелкните на кнопке, и вы увидите, что выполнение Ajax-запроса не сопровождается выводом какой-либо информации о его состоянии.

Настройка базовых параметров Ajax-запросов

Существует группа параметров, с помощью которых можно выполнить базовую настройку Ajax-запроса. Из всех доступных параметров они представляют наименьший интерес, и их имена в основном говорят сами за себя. Параметры, о которых идет речь, приведены в табл. 15.7, и часть из них рассматривается в следующих разделах.

Таблица 15.7. Базовые конфигурационные параметры запроса

Параметр	*Описание*
`accepts`	Устанавливает для запроса значение заголовка `Accept`, который указывает MIME-типы, поддерживаемые браузером. По умолчанию это значение определяется параметром `dataType`
`cache`	Значение `false` указывает на то, что содержимое запроса не должно кэшироваться сервером. По умолчанию кешируются все типы данных, кроме `script` и `jsonp`
`contentType`	Устанавливает для запроса значение заголовка `Content-Type`
`dataType`	Указывает, какие типы данных ожидаются от сервера. Если используется этот параметр, то jQuery будет игнорировать информацию, предоставляемую сервером о типе запроса. Более подробные сведения о том, как работает этот механизм, приведены в главе 14
`headers`	Задает дополнительные заголовки и значения, которые должны включаться в запрос
`jsonp`	Задает строку, которую следует использовать вместо функции обратного вызова при выполнении запросов JSONP. Этот параметр требует согласования с сервером. Более подробная информация о JSONP приведена в главе 14
`jsonpCallback`	Задает имя функции обратного вызова, которое должно использоваться вместо автоматически сгенерированного случайного имени, используемого jQuery по умолчанию
`password`	Задает пароль, который должен использоваться в запросе при прохождении процедуры аутентификации
`scriptCharset`	Указывает jQuery, какой набор символов используется при кодировании запрашиваемого JavaScript-содержимого
`timeout`	Задает длительность тайм-аута (в миллисекундах) для запроса
`userName`	Задает имя пользователя, которое должно использоваться в запросе при прохождении процедуры аутентификации

Задание тайм-аутов и заголовков

О том, что выполняются Ajax-запросы, пользователи часто даже не догадываются, и поэтому указание допустимой длительности тайм-аута — неплохая идея, поскольку это избавит пользователей от томительного ожидания завершения какого-то неведомого для них процесса. Пример задания тайм-аута для запроса приведен в листинге 15.14.

Листинг 15.14. Задание тайм-аута

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
            timeout: 5000,
            headers: {
                "X-HTTP-Method-Override": "PUT"
            },
            success: function(data, status, jqxhr) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            },
            error: function(jqxhr, status, errorMsg) {
                console.log("Ошибка: " + status);
            }
        });
    });
</script>
...
```

В этом примере параметр `timeout` устанавливает максимальную длительность тайм-аута, равную 5 мс. Если запрос за это время не будет выполнен, то вызовется функция, заданная с помощью параметра `error`, и будет выведен код ошибки, определяемый параметром `status`.

Предупреждение. Таймер запускается сразу же после передачи запроса браузеру, и большинство браузеров налагают ограничения на количество одновременно выполняющихся запросов. Это означает, что существует риск того, что к моменту истечения тайм-аута запрос даже не будет запущен. Чтобы избежать этого, необходимо располагать сведениями об ограничениях браузера, а также об объеме и ожидаемой длительности любых других выполняющихся Ajax-запросов.

Дополнительно в этом листинге используется параметр `headers`, с помощью которого в запрос добавляется заголовок, как показано ниже.

```
...
headers: {
    "X-HTTP-Method-Override": "PUT"
},
...
```

Для указания заголовков используется объект отображения данных. Используемый здесь заголовок уже обсуждался ранее. Этот заголовок может быть полезным для создания веб-приложений, поддерживающих архитектурный стиль REST, если только сервер правильно его распознает.

Отправка данных в формате JSON на сервер

Отправка данных в формате JSON на сервер может быть полезна в силу компактности и выразительности этого формата, а также легкости его генерации из объектов JavaScript. Параметр `contentType` позволяет указать для запроса заголовок `Content-Type`, который сообщает серверу тип переданных данных. Пример передачи данных в формате JSON приведен в листинге 15.15.

Листинг 15.15. Отправка данных JSON на сервер

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
            success: function(data, status, jqxhr) {
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            }
        });

        $('button').click(function(e) {
            $.ajax({
                url: $('form').attr("action"),
                contentType: "application/json",
                data: JSON.stringify($('form').serializeArray()),
                type: 'post',
                success: processServerResponse
            })
            e.preventDefault();
        })

        function processServerResponse(data) {
            var inputElems = $('div.dcell').hide();
            for (var prop in data) {
                var filtered = inputElems
                    .has('input[name=' + prop + ']')
                    .appendTo("#row1").show();
            }
            $('#buttonDiv, #totalDiv').remove();
            $('#totalTmpl').tmpl(data).appendTo('body');
        }
    });
</script>
...
```

Здесь параметром `contentType` задается значение `application/json`, которое соответствует MIME-типу формата JSON. Серверу можно было передать любой объект, но я хотел продемонстрировать, как выражаются данные формы в формате JSON.

```
...
data: JSON.stringify($('form').serializeArray()),
...
```

В этой инструкции мы выбираем элемент `form`, в результате чего создается массив объектов, каждый из которых имеет свойства `name` и `value`, представляющие один из элементов `input` формы. С помощью метода `JSON.stringify()` этот массив преобразуется в строку примерно следующего вида.

```
[{"name":"astor","value":"1"}, {"name":"daffodil","value":"1"},
{"name":"rose","value":"1"}, {"name":"peony","value":"1"},
{"name":"primula","value":"1"},{"name":"snowdrop","value":"1"}]
```

В конечном счете мы получаем удобный массив объектов, который можем переслать на сервер. Сценарий сервера Node.js, который используется в этой главе, способен выполнить синтаксический разбор и обработку этого объекта.

Использование дополнительных конфигурационных параметров

В следующих разделах описаны наиболее полезные и заслуживающие внимания дополнительные параметры, применимые к Ajax-запросам. Обычно они редко используются, но в случаях, когда в них возникает потребность, они оказываются незаменимыми. Эти параметры позволяют осуществлять точную настройку взаимодействия jQuery с Ajax.

Создание синхронных запросов

Управление режимом выполнения запросов осуществляется с помощью параметра `async`. Значение `true`, используемое для этого параметра по умолчанию, означает, что запрос будет выполняться в асинхронном режиме, тогда как значению `false` соответствует синхронный режим.

При синхронном выполнении запроса метод `ajax()` ведет себя, как обычная функция, и браузер переходит к выполнению других инструкций сценария лишь после того, как закончится выполнение запроса. Соответствующий пример представлен в листинге 15.16.

Листинг 15.16. Создание синхронного запроса

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var elems;

        $.ajax("flowers.html", {
            async: false,
            success: function(data, status, jqxhr) {
                elems = $(data).filter("div").addClass("dcell");
            }
        });

        elems.slice(0, 3).appendTo('#row1');
        elems.slice(3).appendTo("#row2");
    });
</script>
...
```

Это тот же запрос, который использовался в главе 14 для демонстрации наиболее распространенной ошибки при работе Ajax, но обновленный для использования низкоуровневого API. Данный случай отличается тем, что для параметра `async` установлено значение `false`, и поэтому браузер приступит к выполнению инструкций, в которых вызываются методы `slice()` и `append()`, только после выполнения запроса и присвоения его результата переменной `elem` (в предположении, что запрос будет выполнен успешно). В то же время выполнение синхронных вызовов с использованием метода `ajax()` представляется нелогичным, и поэтому,

прежде чем использовать данную возможность, проанализируйте, нуждаются ли вообще в этом ваши веб-приложения.

Совет. Не используйте синхронные вызовы только лишь потому, что работать с асинхронными вызовами сложнее. Действительно, использование функций обратного вызова и проверка того, что при этом в коде не делаются никакие априорные предположения относительно очередности их выполнения, требует тщательности, однако такой подход стоит затраченных усилий.

Игнорирование данных, оставшихся неизменными

С помощью параметра `ifModified` можно обеспечить получение данных лишь в том случае, если с момента последнего запроса они были изменены. Такое поведение определяется заголовком `Last-Modified`. Благодаря этому удается избежать бесполезной пересылки данных, которая не даст пользователю никакой новой информации по сравнению с той, которой он уже располагает. По умолчанию параметр `ifModified` имеет значение `false`, указывающее jQuery на необходимость игнорирования заголовка `Last-Modified` и предоставления данных в любом случае.

Пример использования этого параметра приведен в листинге 15.17.

Листинг 15.17. Использование параметра `ifModified`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('button').click(function(e) {
            $.ajax("mydata.json", {
                ifModified: true,
                success: function(data, status) {
                    if (status == "success") {
                        $('#row1, #row2').children().remove();
                        var template = $('#flowerTmpl');
                        template.tmpl(data.slice(0, 3))
                            .appendTo("#row1");
                        template.tmpl(data.slice(3))
                            .appendTo("#row2");
                    } else if (status == "notmodified") {
                        $('img')
                            .css("border", "thick solid green");
                    }
                }
            });
            e.preventDefault();
        })
    });
</script>
...
```

В этом примере значение параметра `ifModified` устанавливается равным `true`. Функция `success` вызывается всегда, но если с того момента, когда содержимое запрашивалось в последний раз, оно не изменилось, то аргумент `data` будет иметь значение `undefined`, а аргумент `status` — значение `notmodified`. В данном случае выполняемые действия определяются значением аргумента `status`. Если значени-

ем этого аргумента является success, то аргумент data используется для добавления элементов в документ. Если же аргумент status имеет значение notmodified, то мы используем метод css() для выделения рамкой элементов, которые уже имеются в документе.

В ответ на событие click, связанное с кнопкой, вызывается метод ajax(). Это дает возможность многократно повторять один и тот же запрос, чтобы продемонстрировать влияние параметра ifModified, как показано на рис. 15.3.

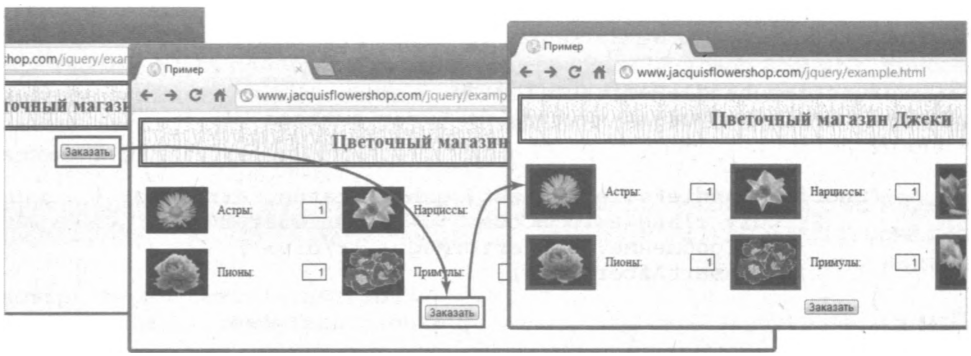


Рис. 15.3. Использование параметра ifModified

Каким бы полезным ни был этот параметр, я рекомендую использовать его с осторожностью. Если отправка запроса является следствием действий пользователя (например, щелчка на кнопке), то существует вероятность того, что пользователь щелкнул на кнопке, поскольку предыдущий запрос не был выполнен так, как ожидалось. Представьте, что вы запрашиваете данные, но в методе, указанном в параметре success, содержится ошибка, которая препятствует правильному обновлению содержимого документа. Тогда вашим следующим действием будет попытка щелкнуть на кнопке еще раз, чтобы добиться ожидаемого результата. Непродуманно используя параметр ifModified, можно проигнорировать действия пользователя, вынуждая его предпринимать более серьезные действия для устранения проблемы.

Обработка кода ответа

Параметр statusCode позволяет выбирать варианты дальнейших действий в зависимости от кода ответов на HTTP-запросы. Его можно использовать либо вместо параметров success и error, либо в дополнение к ним. Пример самостоятельного использования параметра statusCode приведен в листинге 15.18.

Листинг 15.18. Использование параметра statusCode

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "mydata.json",
            statusCode: {
                200: handleSuccessfulRequest,
```

```
                404: handleFailedRequest,
                302: handleRedirect
            }
        });

        function handleSuccessfulRequest(data, status, jqxhr) {
            $('#row1, #row2').children().remove();
            var template = $('#flowerTmpl');
            template.tmpl(data.slice(0, 3)).appendTo("#row1");
            template.tmpl(data.slice(3)).appendTo("#row2");
        }

        function handleRedirect() {
            // эта функция никогда не будет вызвана
        }

        function handleFailedRequest(jqxhr, status, errorMsg) {
            $('<div class=error>Code: ' + jqxhr.status +
                ' Сообщение: ' + errorMsg + '</div>')
                .insertAfter('h1');
        }
    });
</script>
...
```

Здесь параметр `statusCode` задан в виде объекта, устанавливающего связь между кодами ответов на HTTP-запросы и соответствующими им функциями, которые должны быть выполнены на сервере. В данном примере определены три функции, которые соответствуют кодам ответа `200`, `404` и `402`.

Какие именно аргументы передаются функциям[2], зависит от того, отражает ли код ответа успешное выполнение запроса или ошибку. Если код (например, `200`) соответствует успешному запросу, то аргументы совпадают с теми, которые передавались бы функции, определяемой параметром `success`. В противном случае (например, при коде ответа `404`, означающем, что запрашиваемый файл не найден) аргументы совпадают с теми, которые передавались бы функции, определяемой параметром `error`.

Обратите внимание на то, что функция задана и для кода ответа `302`. Этот код отправляется обратно браузеру, если сервер намерен перенаправить запрос по другому URL-адресу. Библиотека jQuery автоматически следует инструкциям перенаправления до тех пор, пока не получит некоторое содержимое или не столкнется с ошибкой. В данном случае это означает, что функция, указанная для кода `302`, вообще не будет вызываться.

Совет. Код ответа `304`, указывающий на то, что с момента последнего запроса содержимое не изменилось, генерируется, лишь если используется параметр `ifmodified`. В противном случае jQuery отправляет вместо него код `200`. Параметр `ifmodified` описан в предыдущем разделе.

Как видите, это средство не дает непосредственной информации о кодах ответа. Я часто пользуюсь им в процессе отладки взаимодействия браузера с сервером, обычно для того, чтобы выяснить, почему jQuery ведет себя не так, как мне хотелось бы. При этом я использую параметр `statusCode` в дополнение к параметрам `success` и `error` и вывожу информацию на консоль. Если эти параметры исполь-

---

[2] Здесь имеется в виду сигнатура функции. — *Примеч. ред.*

зуются совместно, то сначала будут выполнены функции success и error, а затем уже — функции, определяемые параметром statusCode.

Предварительная очистка ответных данных

С помощью параметра dataFilter можно задать функцию, которая будет вызвана для предварительной очистки данных, возвращаемых сервером. Это средство незаменимо в тех случаях, когда пересылаемые сервером данные не совсем вас устраивают либо из-за того, что отформатированы неподходящим образом, либо из-за того, что среди них есть данные, которые вы не хотите обрабатывать. Это средство мне очень помогает при работе с серверами Microsoft ASP.NET, присоединяющими лишние данные к данным JSON. Удаление таких данных с помощью параметра dataFilter требует лишь минимальных усилий. Пример использования параметра dataFilter приведен в листинге 15.19.

Листинг 15.19. Использование параметра dataFilter

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "mydata.json",
            success: function(data, status, jqxhr) {
                $('#row1, #row2').children().remove();
                var template = $('#flowerTmpl');
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            },
            dataType: "json",
            dataFilter: function(data, dataType) {
                if (dataType == "json") {
                    var filteredData = $.parseJSON(data);
                    filteredData.shift();
                    return JSON.stringify(filteredData.reverse());
                } else {
                    return data;
                }
            }
        });
    });
</script>
...
```

Функции передаются данные, полученные с сервера, и значение параметра dataType. Если параметр dataType не используется, то второму аргументу присваивается значение undefined. Ваша задача заключается в том, чтобы вернуть отфильтрованные данные. В этом примере предмет нашего внимания — данные в формате JSON.

```
...
var filteredData = $.parseJSON(data);
filteredData.shift();
return JSON.stringify(filteredData.reverse());
...
```

Для повышения иллюстративности примера в нем выполняются некоторые дополнительные операции. Во-первых, данные JSON преобразуются в массив JavaScript с помощью метода jQuery `parseJSON` (один из вспомогательных методов jQuery, описанных в главе 33). Затем из массива удаляется первый элемент с помощью метода `shift()`, а порядок следования остальных его элементов обращается с помощью метода `reverse()`.

Все, что требуется от функции, — вернуть строку, и поэтому мы вызываем метод `JSON.stringify()`, зная, что jQuery преобразует данные в объект JavaScript, прежде чем вызвать функцию `success`. В данном примере была продемонстрирована возможность удаления элемента из массива, однако, в зависимости от ситуации, мы могли бы выполнить любой другой вид обработки. Конечный результат представлен на рис. 15.4.

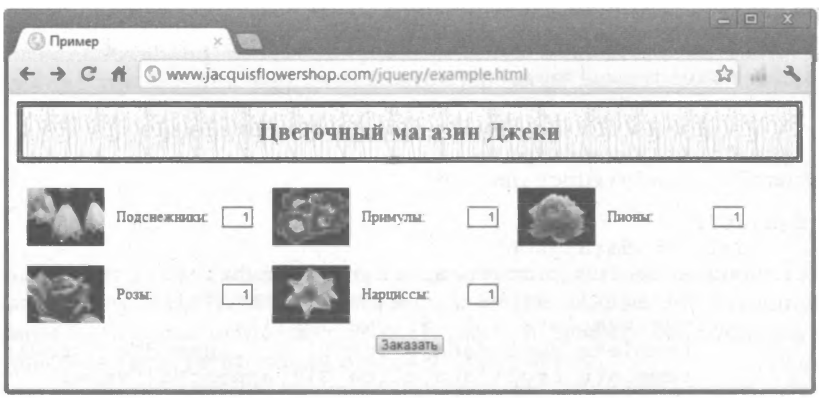


Рис. 15.4. Удаление элемента и изменение порядка следования данных с помощью параметра `dataFilter`

Управление преобразованием данных

Рассмотрение одной из самых любимых своих настроек я приберег напоследок. Должно быть, вы обратили внимание, что при получении определенных типов данных jQuery автоматически выполняет некоторые удобные преобразования. Например, получая данные JSON, jQuery предоставляет функцию `success`, использующую объект JavaScript, а не исходную необработанную строку JSON.

Для управления подобными преобразованиями используется параметр `converters`. Значением этого параметра является объект, устанавливающий соответствие между типами данных и функциями, используемыми для их обработки. В листинге 15.20 показано, как использовать этот параметр для автоматического преобразования HTML-данных в объект jQuery.

Листинг 15.20. Использование параметра `converters`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "flowers.html",
```

```
            success: function(data, status, jqxhr) {
                var elems = data.filter('div').addClass("dcell");
                elems.slice(0, 3).appendTo('#row1');
                elems.slice(3).appendTo("#row2");
            },
            converters: {
                "text html": function(data) {
                    return $(data);
                }
            }
        });
    });
</script>
...
```

В этом примере регистрируется функция для типа данных `text html`. Обратите внимание на пробел между компонентами указываемого MIME-типа (в отличие от формы записи `text/html`). Функция принимает данные, полученные от сервера, и возвращает преобразованные данные. В этом случае преобразование данных заключается в передаче HTML-фрагмента, содержащегося в файле `flowers.html`, функции `$()` и возврате результата. Отсюда следует, что к объекту, передаваемому в качестве аргумента `data` функции `success`, применимы обычные методы jQuery.

Совет. Типы данных не всегда совпадают с MIME-типами, возвращаемыми сервером. Например, MIME-тип `application/json` обычно предоставляется методу `converters` как `"text html"`.

Работая с преобразователями данных, можно слишком увлечься. Я всегда стараюсь избегать соблазна делать с помощью этих функций больше, чем следует. Например, иногда меня так и тянет применить к данным JSON шаблон и передать обратно полученные в результате HTML-элементы. И хотя этот прием очень удобен, он сослужит плохую службу, если кто-то другой будет пытаться расширить ваш код или, например, вам самому впоследствии потребуется организовать интенсивную обработку данных для получения их в исходной форме.

Настройка и фильтрация Ajax-запросов

После того как вы познакомились с методом `ajax()` и доступными для работы с ним параметрами, мы можем рассмотреть несколько дополнительных методов, предоставляемых jQuery для упрощения настройки запросов.

Определение параметров, используемых по умолчанию

Метод `ajaxSetup()` позволяет установить значения параметров, которые будут применяться по умолчанию во всех Ajax-запросах, тем самым освобождая вас от необходимости настраивать параметры при каждом запросе. Пример использования этого метода приведен в листинге 15.21.

Листинг 15.21. Использование метода `ajaxSetup()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $.ajaxSetup({
            timeout: 15000,
            global: false,
            error: function(jqxhr, status, errorMsg) {
                $('<div class=error/>')
                    .text("Status: " + status + " Error: " +
                        errorMsg)
                    .insertAfter('h1');
            },
            converters: {
                "text html": function(data) {
                    return $(data);
                }
            }
        });

        $.ajax({
            url: "flowers.html",
            success: function(data, status, jqxhr) {
                var elems = data.filter('div').addClass("dcell");
                elems.slice(0, 3).appendTo('#row1');
                elems.slice(3).appendTo("#row2");
            },
        });
    });
</script>
...
```

Метод `ajaxSetup()` вызывается с помощью функции jQuery `$` аналогично тому, как это делалось в случае вызова метода `ajax()`. Аргументом метода `ajaxSetup()` является объект, содержащий значения параметров, которые вы хотите использовать по умолчанию для всех Ajax-запросов. В этом примере мы устанавливаем значения по умолчанию для параметров `timeout`, `global`, `error` и `converters`. После того как был вызван метод `ajaxSetup()`, нам остается определить значения лишь те значения параметров, которые мы хотим изменить, или те, которые не предоставляются по умолчанию. Это обеспечивает значительную экономию времени в тех случаях, когда приходится делать множество запросов с одинаковыми значениями параметров.

Совет. Указанные при вызове метода `ajaxSetup()` параметры применяются также к запросам, выполняемым с помощью прямых и вспомогательных методов, описанных в главе 14. Это позволяет замечательным образом сочетать возможности детального управления, обеспечиваемого низкоуровневым API, с простотой использования вспомогательных методов.

Фильтрация запросов

Метод `ajaxSetup()` определяет базовые значения конфигурационных параметров, применимые ко всем запросам Ajax. Возможности динамической настройки параметров для отдельных Ajax-запросов обеспечиваются методом `ajaxPrefilter()`. Пример использования этого метода приведен в листинге 15.22.

Листинг 15.22. Использование метода `ajaxPrefilter()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajaxSetup({
            timeout: 15000,
            global: false,
            error: function(jqxhr, status, errorMsg) {
                $('<div class=error/>')
                    .text("Status: " + status + " Error: " +
                        errorMsg)
                    .insertAfter('h1');
            },
            converters: {
                "text html": function(data) {
                    return $(data);
                }
            }
        })

        $.ajaxPrefilter("json html", function(settings,
                originalSettings, jqxhr) {
            if (originalSettings.dataType == "html") {
                settings.timeout = 2000;
            } else {
                jqxhr.abort();
            }
        })

        $.ajax({
            url: "flowers.html",
            dataType: "html",
            success: function(data, status, jqxhr) {
                var elems = data.filter('div').addClass("dcell");
                elems.slice(0, 3).appendTo('#row1');
                elems.slice(3).appendTo("#row2");
            },
        });
    });
</script>
...
```

Указанная вами функция будет выполняться для каждого нового Ajax-запроса. Аргументами, передаваемыми функции, являются параметры запроса (включая любые значения по умолчанию, установленные вами с помощью метода `ajaxSetup()`), а также исходные параметры, переданные методу `ajax()` (исключая любые значения по умолчанию) и объекту `jqXHR` запроса. Мы вносим изменения в объект, передаваемый в качестве первого аргумента, как показано в примере. В данном сценарии, если среди параметров, передаваемых методу `ajax()`, присутствует параметр `dataType`, то длительность тайм-аута устанавливается равной двум секундам. Чтобы предотвратить отправку всех остальных запросов, для объекта `jqXHR` вызывается метод `abort()`.

Резюме

В этой главе вы познакомились с низкоуровневым программным интерфейсом Ajax, поддерживаемым библиотекой jQuery. Работать с ним не сложнее, чем с прямыми или вспомогательными методами, описанными в главе 14 (надеюсь, вы с этим согласитесь). Ценой очень незначительных усилий вы получаете возможность контролировать многие аспекты обработки Ajax-запросов и настраивать параметры этого процесса в точном соответствии с вашими потребностями и предпочтениями.

ГЛАВА 16

Рефакторинг примера (часть II)

В этой части книги вы познакомились с целым рядом весьма эффективных средств, и, как и в конце предыдущей части, я хочу объединить их в одном примере, чтобы перед вами предстала цельная картина возможностей jQuery. Поддерживать структуру документа в таком виде, чтобы он мог отображаться даже в браузерах, в которых выполнение сценариев JavaScript запрещено, я не буду, поскольку в этой главе все средства, добавляемые в базовый пример, существенно зависят от использования JavaScript.

Пересмотр переработанного варианта примера

В главе 11 мы использовали основные возможности jQuery для улучшения образца документа путем включения в него DOM-манипуляций, эффектов и событий. Полученный в результате документ, который послужит отправной точкой в этой главе и будет дополнен возможностями, обсуждавшимися в данной части, приведен в листинге 16.1.

В сценарии имеется ряд мест, в которых вставка элементов в документ осуществляется динамическим способом. Я не делаю эти элементы статическими и оставляю все как есть, чтобы ваше внимание было сосредоточено исключительно на улучшении документа за счет добавления в него новых возможностей.

Листинг 16.1. Исходный документ для этой главы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        a.arrowButton {
            background-image: url(leftarrows.png); float: left;
            margin-top: 15px; display: block; width: 50px;
            height: 50px;
        }
        #right {background-image: url(rightarrows.png)}
        h1 { min-width: 0px; width: 95%; }
        #oblock { float: left; display: inline;
            border: thin black solid; }
```

```
        form { margin-left: auto; margin-right: auto;
            width: 885px; }
        #bbox {clear: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            var fNames = ["Carnation", "Lily", "Orchid"];
            var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
            var fRow = $('<div id=row3 class=drow/>')
                .appendTo('div.dtable');
            var fTemplate = $('<div class=dcell><img/><label/>
                <input/></div>');
            for (var i = 0; i < fNames.length; i++) {
                fTemplate.clone().appendTo(fRow).children()
                    .filter('img').attr('src', fNames[i] + ".png")
                    .end()
                    .filter('label').attr('for', fNames[i])
                    .text(frNames[i]).end()
                    .filter('input').attr({name: fNames[i],
                        value: 0, required: "required"})
            }

            $('<a id=left></a><a id=right></a>')
                .prependTo('form').addClass("arrowButton")
                .click(handleArrowPress).hover(handleArrowMouse);
            $('#right').appendTo('form');

            $('#row2, #row3').hide();

            var total = $('#buttonDiv')
                .prepend("<div>Всего заказано: <span id=total>0
                    </span></div>")
                .css({clear: "both", padding: "5px"});
            $('<div id=bbox />').appendTo("body").append(total);

            $('input').change(function(e) {
                var total = 0;
                $('input').each(function(index, elem) {
                    total += Number($(elem).val());
                });
                $('#total').text(total);
            });

            function handleArrowMouse(e) {
               var propValue = e.type == "mouseenter" ?
                   "-50px 0px" : "0px 0px";
               $(this).css("background-position", propValue);
            }

            function handleArrowPress(e) {
                var elemSequence = ["row1", "row2", "row3"];

                var visibleRow = $('div.drow:visible');
                var visibleRowIndex = jQuery
                    .inArray(visibleRow.attr("id"),elemSequence);

                var targetRowIndex;

                if (e.target.id == "left") {
```

```
                targetRowIndex = visibleRowIndex - 1;
                if (targetRowIndex < 0)
                    {targetRowIndex = elemSequence.length -1};
            } else {
                targetRowIndex = (visibleRowIndex + 1) %
                    elemSequence.length;
            }
            visibleRow.fadeOut("fast", function() {
                $('#' + elemSequence[targetRowIndex])
                    .fadeIn("fast")});
        }
    });
  </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <img src="astor.png"/>
                        <label for="astor">Астры:</label>
                        <input name="astor" value="0" />
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0"/>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" />
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" />
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" />
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" />
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
```

```
        </div>
    </form>
</body>
</html>
```

Этот документ не совсем совпадает с тем документом, которым заканчивается глава 11. В текущем варианте документа исключены многочисленные вызовы метода `css()` для отдельных наборов выбранных элементов, вместо которых теперь используется элемент `style`. Вид исправленного документа (разумеется, с учетом тех изменений, которые были внесены в главе 11) приведен на рис. 16.1.

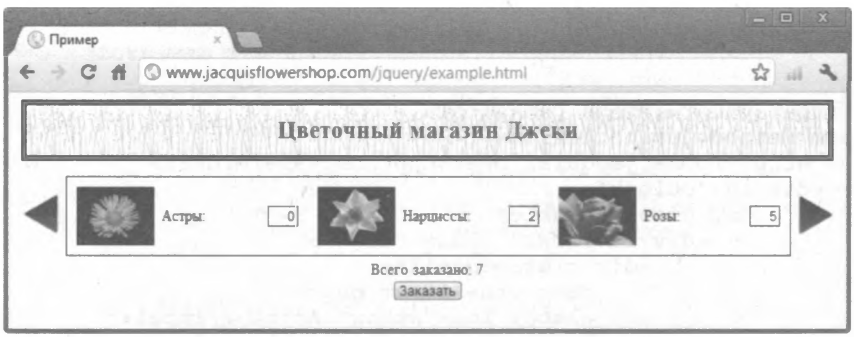


Рис. 16.1. Исходный образец документа для этой главы

Обновление сценария для сервера Node.js

Прежде чем приступить к дальнейшей работе с jQuery, нам потребуется обновить сценарий серверной стороны. Изменения, которые мы в него внесем, расширят состав данных, возвращаемых сервером после отправки формы, и обеспечат проверку корректности данных, введенных пользователем. Обновленный вариант сценария представлен в листинге 16.2.

Листинг 16.2. Пересмотренный вариант сценария для сервера Node.js

```
var http = require('http');
var url = require('url');
var querystring = require('querystring');
http.createServer(function (req, res) {
    console.log("Request: " + req.method +
        " to " + req.url);

    if (req.method == 'OPTIONS') {
        res.writeHead(200, "OK", {
            "Access-Control-Allow-Headers": "Content-Type",
            "Access-Control-Allow-Methods": "*",
            "Access-Control-Allow-Origin": "*"
            });
        res.end();

    } else if (req.method == 'POST') {
        var dataObj = new Object();
        var contentType = req.headers["content-type"];
```

```
        var fullBody = '';

        if (contentType) {
            if (contentType.indexOf("application/x-www-form-
                urlencoded") > -1) {
                req.on('data', function(chunk) { fullBody +=
                    chunk.toString();});
                req.on('end', function() {
                    var dBody = querystring.parse(fullBody);
                    writeResponse(req, res, dBody,
                        url.parse(req.url, true)
                            .query["callback"])
                });
            } else {
                req.on('data', function(chunk) { fullBody +=
                    chunk.toString();});
                req.on('end', function() {
                    dataObj = JSON.parse(fullBody);
                    var dprops = new Object();
                    for (var i = 0; i < dataObj.length; i++) {
                        dprops[dataObj[i].name] = dataObj[i]
                            .value;
                    }
                    writeResponse(req, res, dprops);
                });
            }
        }
    } else if (req.method == "GET") {
        var data = url.parse(req.url, true).query;
        writeResponse(req, res, data, data["callback"])
    }

}).listen(9999);
console.log("Ready on post 9999");

var flowerData = {
    astor: { price: 2.99, stock: 10, plural: "астр"},
    daffodil: {price: 1.99, stock: 10, plural: "нарциссов"},
    rose: {price: 4.99, stock: 2, plural: "роз"},
    peony: {price: 1.50, stock: 3, plural: "пионов"},
    primula: {price: 3.12, stock: 20, plural: "примул"},
    snowdrop: {price: 0.99, stock: 5, plural: "подснежников"},
    carnation: {price: 0.50, stock: 1, plural: "гвоздик"},
    lily: {price: 1.20, stock: 2, plural: "лилий"},
    orchid: {price: 10.99, stock: 5, plural: "орхидей"}
}

function writeResponse(req, res, data, jsonp) {
    var jsonData;
    if (req.url == "/stockcheck") {
        for (flower in data) {
            if (flowerData[flower].stock >= data[flower]) {
                jsonData = true;
            } else {
                jsonData = "В наличии имеется только " +
                    flowerData[flower].stock + " "
                    + flowerData[flower].plural;
            }
```

```
            break;
        }
        jsonData = JSON.stringify(jsonData);
    } else {
        var totalCount = 0;
        var totalPrice = 0;
        for (item in data) {
            if(item != "_" && data[item] > 0) {
                var itemNum = Number(data[item])
                totalCount += itemNum;
                totalPrice += (itemNum * flowerData[item].price);
            } else {
                delete data[item];
            }
        }
        data.totalItems = totalCount;
        data.totalPrice = totalPrice.toFixed(2);

        jsonData = JSON.stringify(data);
        if (jsonp) {
            jsonData = jsonp + "(" + jsonData + ")";
        }
    }
    res.writeHead(200, "OK", {
        "Content-Type": jsonp ?
            "text/javascript" : "application/json",
        "Access-Control-Allow-Origin": "*"});
    res.write(jsonData);
    res.end();
}
```

Теперь ответ браузеру включает в себя общую стоимость товарных единиц, выбранных с помощью элементов формы и переданных серверу, причем результат, возвращаемый в формате JSON, будет иметь примерно следующий вид.

```
{"astor":"1","daffodil":"2","rose":"4","totalItems":7,
"totalPrice":"26.93"}
```

Я сохранил этот сценарий в файле `formserver.js`. Самый простой способ получить его — загрузить архив примеров на сайте книги (см. главу 1). Для запуска сценария введите в командной строке следующую команду.

```
node.exe formserver.js
```

Подготовка к работе с Ajax

Для начала добавим некоторые базовые элементы и стили, которые будут использоваться для отображения информации об ошибках, связанных с выполнением Ajax-запросов, а также настроим базовые параметры, которые будут применяться ко всем Ajax-запросам. Соответствующие изменения, вносимые в документ, представлены в листинге 16.3.

Листинг 16.3. Настройка поддержки запросов Ajax и обработки ошибок

```
...
<style type="text/css">
    a.arrowButton {
        background-image: url(leftarrows.png); float: left;
        margin-top: 15px; display: block; width: 50px;
            height: 50px;}
    #right {background-image: url(rightarrows.png)}
    h1 { min-width: 0px; width: 95%; }
    #oblock { float: left; display: inline;
        border: thin black solid; }
    form { margin-left: auto; margin-right: auto; width: 885px; }
    #bbox {clear: left}
    #error {color: red; border: medium solid red; padding: 4px;
        margin: auto; width: 300px;text-align: center;
        margin-bottom: 5px}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $.ajaxSetup({
            timeout: 5000,
            converters: {
                "text html": function(data) {
                    return $(data);
                }
            }
        })

        $(document)
            .ajaxError(function(e, jqxhr, settings, errorMsg) {
            $('#error').remove();
            var msg = "Произошла ошибка. Пожалуйста,
                       повторите запрос"
            if (errorMsg == "timeout") {
                msg = "Время запроса истекло. Пожалуйста,
                       повторите запрос"
            } else if (jqxhr.status == 404) {
                    msg = "Файл не найден";
            }
            $('<div id=error/>').text(msg).insertAfter('h1');
        }).ajaxSuccess(function() {
            $('#error').remove();
        })

        var fNames = ["Carnation", "Lily", "Orchid"];
        var frNames = ["Гвоздики", "Лилии", "Орхидеи"];
        var fRow = $('<div id=row3 class=drow/>')
            .appendTo('div.dtable');
        var fTemplate = $('<div class=dcell><img/><label/><input/>
            </div>');
        for (var i = 0; i < fNames.length; i++) {
            fTemplate.clone().appendTo(fRow).children()
                .filter('img').attr('src', fNames[i] +
```

```
                    ".png").end()
                .filter('label').attr('for', fNames[i])
                    .text(frNames[i]).end()
                .filter('input').attr({name: fNames[i],
                                       value: 0, required: "required"})
        }

        $('<a id=left></a><a id=right></a>').prependTo('form')
            .addClass("arrowButton").click(handleArrowPress)
            .hover(handleArrowMouse);
        $('#right').appendTo('form');

        $('#row2, #row3').hide();

        var total = $('#buttonDiv')
            .prepend("<div>Total Items: <span id=total>0</span>
                </div>")
            .css({clear: "both", padding: "5px"});
        $('<div id=bbox />').appendTo("body").append(total);

        $('input').change(function(e) {
            var total = 0;
            $('input').each(function(index, elem) {
                total += Number($(elem).val());
            });
            $('#total').text(total);
        });

        function handleArrowMouse(e) {
           var propValue = e.type == "mouseenter" ?
               "-50px 0px" : "0px 0px";
           $(this).css("background-position", propValue);
        }

        function handleArrowPress(e) {
            var elemSequence = ["row1", "row2", "row3"];

            var visibleRow = $('div.drow:visible');
            var visibleRowIndex = jQuery
                .inArray(visibleRow.attr("id"),elemSequence);

            var targetRowIndex;

            if (e.target.id == "left") {
                targetRowIndex = visibleRowIndex - 1;
                if (targetRowIndex < 0)
                    {targetRowIndex = elemSequence.length -1};
            } else {
                targetRowIndex = (visibleRowIndex + 1) %
                    elemSequence.length;
            }
            visibleRow.fadeOut("fast", function() {
                $('#' + elemSequence[targetRowIndex])
                    .fadeIn("fast")});
        }
    });
</script>
...
```

Для настройки простого отображения сообщений об ошибках здесь используются глобальные события Ajax. Если происходит ошибка, на экран дополнительно выводятся новые элементы, содержащие описание проблемы. Выводимые сообщения создаются на основе информации, получаемой от jQuery, но организованы на очень простом уровне. В реальных веб-приложениях эти сообщения должны быть более информативными и по возможности должны содержать рекомендации по устранению проблем. Благодаря глобальным сообщениям появилась возможность применить параметры `success` и `error` к отдельным запросам, не утруждая себя объединением многочисленных функций. Пример вывода простого сообщения об ошибке приведен на рис. 16.2.

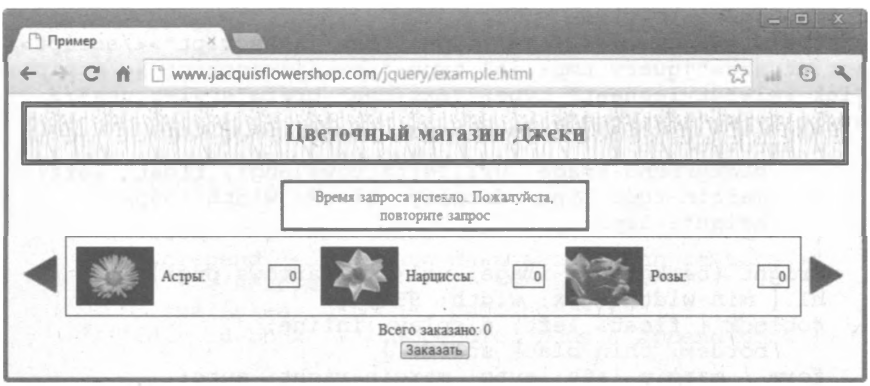


Рис. 16.2. Отображение сообщения об ошибке при выполнении Ajax-запроса

Сообщение остается на экране до тех пор, пока не будет выполнен успешный запрос или не произойдет другая ошибка, и тогда элементы будут удалены из документа.

Кроме событий, в сценарии используется метод `ajax()`, который определяет значения параметра `timeout` и предоставляет конвертер для HTML-фрагментов, что обеспечивает их автоматическую обработку средствами jQuery.

Вынесение информации о продукции в отдельный файл

Следующее изменение, которое мы собираемся внести, касается удаления из документа существующих элементов, соответствующих различным видам цветочной продукции, и цикла, с помощью которого в список добавляются три дополнительных вида цветов. Мы заменим их внешним шаблоном данных и вставим в сценарий пару вызовов Ajax. Для этого я создал файл под названием `additionalflowers.json`, содержимое которого представлено в листинге 16.4.

Листинг 16.4. Содержимое файла `additionalflowers.json`

```
[{"name":"Гвоздики","product":"carnation"},
 {"name":"Лилии","product":"lily"},
 {"name":"Орхидеи","product":"orchid"}]
```

В этом файле содержится простое описание в формате JSON дополнительных видов цветов, которые мы хотим отобразить. Мы получим основной набор элементов с описаниями цветов в виде HTML-фрагмента, а затем добавим в него новые элементы, обрабатывая данные JSON. Указанные изменения показаны в листинге 16.5.

Листинг 16.5. Формирование состава отображаемой продукции путем использования HTML-фрагмента и данных JSON, полученных посредством Ajax-запроса

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        a.arrowButton {
            background-image: url(leftarrows.png); float: left;
            margin-top: 15px; display: block; width: 50px;
            height: 50px;
        }
        #right {background-image: url(rightarrows.png)}
        h1 { min-width: 0px; width: 95%; }
        #oblock { float: left; display: inline;
            border: thin black solid; }
        form { margin-left: auto; margin-right: auto;
            width: 885px; }
        #bbox {clear: left}
        #error {color: red; border: medium solid red;
        padding: 4px; margin: auto; width: 300px;
        text-align: center; margin-bottom: 5px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $.ajaxSetup({
                timeout: 5000,
                converters: {
                    "text html": function(data)
                        { return $(data); }
                }
            })

            $(document)
                    .ajaxError(function(e, jqxhr,
                     settings, errorMsg) {
                $('#error').remove();
                var msg = "Произошла ошибка. Пожалуйста,
                          повторите запрос"
                if (errorMsg == "timeout") {
                    msg = "Время запроса истекло. Пожалуйста,
                          повторите запрос"
} else if (jqxhr.status == 404) {
                        msg = "Файл не найден";
                }
                $('<div id=error/>').text(msg).insertAfter('h1');
            }).ajaxSuccess(function() {
```

```
    $('#error').remove();
})

$('#row2, #row3').hide();

$.get("flowers.html", function(data) {
    var elems = data.filter('div').addClass("dcell");
    elems.slice(0, 3).appendTo('#row1');
    elems.slice(3).appendTo("#row2");
})

$.getJSON("additionalflowers.json", function(data) {
    $('#flowerTmpl').tmpl(data).appendTo("#row3");
})

$('<a id=left></a><a id=right></a>').prependTo('form')
    .addClass("arrowButton").click(handleArrowPress)
    .hover(handleArrowMouse);
$('#right').appendTo('form');

var total = $('#buttonDiv')
    .prepend("<div>Всего заказано: <span id=total>0
        </span></div>")
    .css({clear: "both", padding: "5px"});
$('<div id=bbox />').appendTo("body").append(total);

$('input').change(function(e) {
    var total = 0;
    $('input').each(function(index, elem) {
        total += Number($(elem).val());
    });
    $('#total').text(total);
});

function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ?
        "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}

function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];

    var visibleRow = $('div.drow:visible');
    var visibleRowIndex = jQuery
        .inArray(visibleRow.attr("id"),elemSequence);

    var targetRowIndex;

    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0)
            {targetRowIndex = elemSequence.length -1};
    } else {
        targetRowIndex = (visibleRowIndex + 1) %
            elemSequence.length;
    }
```

```
                visibleRow.fadeOut("fast", function() {
                    $('#' + elemSequence[targetRowIndex])
                    .fadeIn("fast")});
            }
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
                <div id="row3" class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv"><button type="submit">Заказать
            </button></div>
    </form>
</body>
</html>
```

Для получения HTML-фрагмента и данных JSON, которые нужны нам для создания рядов в табличной компоновке страницы, в сценарии используются прямые методы Ajax. Здесь уместно отметить одну приятную особенность прямых методов. Поскольку они представляют собой оболочки, служащие для вызова функций низкоуровневого API с заранее установленными значениями ряда параметров, то все настройки параметров, выполненные с помощью метода `ajaxSetup()`, применимы к ним в той же степени, как если бы они применялись непосредственно к методу `ajax()`.

Кроме вызовов методов `get()` и `getJSON()`, сценарий дополнен простым шаблоном, облегчающим и упрощающим обработку данных JSON. Внешний вид документа остался прежним, изменился лишь источник его содержимого.

Добавление проверки данных формы

Следующий этап заключается в добавлении возможности проверки данных, введенных в полях формы. Соответствующие изменения представлены в листинге 16.6.

Листинг 16.6. Добавление проверки данных формы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
```

```
<script src="jquery-1.7.js" type="text/javascript"></script>
<script src="jquery.tmpl.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
    a.arrowButton {
        background-image: url(leftarrows.png); float: left;
        margin-top: 15px; display: block; width: 50px;
        height: 50px;
    }
    #right {background-image: url(rightarrows.png)}
    h1 { min-width: 0px; width: 95%; }
    #oblock { float: left; display: inline;
        border: thin black solid; }
    form { margin-left: auto; margin-right: auto;
        width: 885px; }
    #bbox {clear: left}
    #error {color: red; border: medium solid red;
        padding: 4px; margin: auto; width: 300px;
        text-align: center; margin-bottom: 5px}
    .invalidElem {border: medium solid red}
    #errorSummary {border: thick solid red; color: red;
        width: 350px; margin: auto; padding: 4px;
        margin-bottom: 5px}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $.ajaxSetup({
            timeout: 5000,
            converters: {
                "text html": function(data)
                    { return $(data); }
            }
        })

        $(document)
                .ajaxError(function(e, jqxhr,
                 settings, errorMsg) {
            $('#error').remove();
            var msg = "Произошла ошибка. Пожалуйста,
                      повторите запрос"
            if (errorMsg == "timeout") {
                msg = "Время запроса истекло. Пожалуйста,
                      повторите запрос"
            } else if (jqxhr.status == 404) {
                    msg = "Файл не найден";
            }
            $('<div id=error/>').text(msg).insertAfter('h1');
        }).ajaxSuccess(function() {
            $('#error').remove();
        })

        $('#row2, #row3').hide();

        var flowerReq = $.get("flowers.html", function(data) {
            var elems = data.filter('div').addClass("dcell");
            elems.slice(0, 3).appendTo('#row1');
```

```
        elems.slice(3).appendTo("#row2");
    })

    var jsonReq = $.getJSON("additionalflowers.json",
        function(data) {
        $('#flowerTmpl').tmpl(data).appendTo("#row3");
    })

    $('<div id=errorSummary>Пожалуйста, исправьте
        следующие ошибки:</div>')
        .append('<ul id="errorsList"></ul>')
        .hide().insertAfter('h1');

    $('form').validate({
        highlight: function(element, errorClass) {
            $(element).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).removeClass("invalidElem");
        },
        errorContainer: '#errorSummary',
        errorLabelContainer: '#errorsList',
        wrapper: 'li',
        errorElement: "div"
    });

    var plurals = {
        astor: "астр", daffodil: "нарциссов", rose: "роз",
        peony: "пионов", primula: "примул",
        snowdrop: "подснежников", carnation: "гвоздик",
        lily: "лилий", orchid: "орхидей"
    }

    $.when(flowerReq, jsonReq).then(function() {
        $('input').each(function(index, elem) {
            $(elem).rules("add", {
                required: true,
                min: 0,
                digits: true,

                messages: {
                  required: "Пожалуйста, введите
                      количество " + plurals[elem.name],
                  digits: "Пожалуйста, введите количество
                      " + plurals[elem.name],
                  min: "Пожалуйста, введите положительное
                      число для " + plurals[elem.name]
                }
            })
        }).change(function(e) {
            if ($('form').validate()
                .element($(e.target))) {
                var total = 0;
                $('input').each(function(index, elem) {
                    total += Number($(elem).val());
```

```
                });
                $('#total').text(total);
            }
        });
    });
    $('<a id=left></a><a id=right></a>').prependTo('form')
        .addClass("arrowButton").click(handleArrowPress)
        .hover(handleArrowMouse);
    $('#right').appendTo('form');

    var total = $('#buttonDiv')
        .prepend("<div>Всего заказано: <span id=total>0
            </span></div>")
        .css({clear: "both", padding: "5px"});
    $('<div id=bbox />').appendTo("body").append(total);

    $('input').change(function(e) {
        var total = 0;
        $('input').each(function(index, elem) {
            total += Number($(elem).val());
        });
        $('#total').text(total);
    });

    function handleArrowMouse(e) {
        var propValue = e.type == "mouseenter" ?
            "-50px 0px" : "0px 0px";
        $(this).css("background-position", propValue);
    }

    function handleArrowPress(e) {
        var elemSequence = ["row1", "row2", "row3"];

        var visibleRow = $('div.drow:visible');
        var visibleRowIndex = jQuery
            .inArray(visibleRow.attr("id"),elemSequence);

        var targetRowIndex;

        if (e.target.id == "left") {
            targetRowIndex = visibleRowIndex - 1;
            if (targetRowIndex < 0)
                {targetRowIndex = elemSequence.length -1};
        } else {
            targetRowIndex = (visibleRowIndex + 1) %
                elemSequence.length;
        }
        visibleRow.fadeOut("fast", function() {
            $('#' + elemSequence[targetRowIndex])
            .fadeIn("fast")});
    }
});
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
```

```
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
                <div id="row3" class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv"><button type="submit">Заказать
            </button></div>
    </form>
</body>
</html>
```

В этом листинге импортируется библиотека JavaScript для подключаемого модуля проверки данных формы и определяются некоторые базовые стили, которые будут использоваться для вывода сообщений об обнаруженных в процессе проверки ошибках. Далее мы настраиваем процедуру проверки путем вызова метода `validate()` для элемента `form`, устанавливая при этом создание сводного отчета об ошибках. Этот подход в точности соответствует тому, который обсуждался в главе 13.

Вместе с тем использование Ajax для генерации элементов, соответствующих отдельным видам цветов, ставит перед нами одну проблему. Разумеется, нам предстоит иметь дело с асинхронными вызовами, и поэтому в инструкциях, которые следуют за Ajax-вызовами, мы не можем делать никаких предположений относительно наличия в документе элементов `input`. Многие программисты часто попадают в эту ловушку, о чем уже говорилось в главе 14, и если браузер приступит к обработке выбираемого набора элементов `input` еще до того, как завершатся Ajax-запросы, то окажется, что выбирать нечего (поскольку элементы еще не были созданы и включены в документ), и проверка не сможет быть выполнена. Чтобы обойти это затруднение, в сценарии используются методы `when()` и `then()`, входящие в число средств, предусмотренных для отсроченных объектов jQuery, которые описываются в главе 35. Соответствующий код приводится ниже.

```
...
$.when(flowerReq, jsonReq).then(function() {
    $('input').each(function(index, elem) {
        $(elem).rules("add", {
            required: true,
            min: 0,
            digits: true,
            messages: {
                required: "Пожалуйста, введите
                          количество " + plurals[elem.name],
                digits: "Пожалуйста, введите количество " +
                            plurals[elem.name],
                min: "Пожалуйста, введите положительное
                      число для " + plurals[elem.name]
                }
```

```
            })
        }).change(function(e) {
            if ($('form').validate()
                    .element($(e.target))) {
                var total = 0;
                $('input').each(function(index, elem) {
                    total += Number($(elem).val());
                });
                $('#total').text(total);
            }
    });
});
...
```

Я не хочу забегать далеко вперед и лишь отмечу, что объекты `jqXHR`, которые возвращаются всеми методами Ajax, можно передавать в качестве аргументов методу `when()`, и если оба запроса окажутся успешными, то будет выполнена функция, переданная методу `then()`.

Настройка проверки данных формы осуществляется в функции, передаваемой методу `then()`, путем выбора элементов `input` и добавления правил проверки, соответствующих каждому из них. Согласно этим правилам поля являются обязательными для заполнения, значения должны быть представлены цифрами и допустимым минимальным значением является нулевое значение. Для каждой контрольной проверки определено свое сообщение. Для повышения содержательности сообщений в них включаются данные из массива, содержащего названия цветов в родительном падеже множественного числа.

Выбрав элементы `input` формы, мы получаем возможность предоставить функцию-обработчик для события `change` при изменении значения, введенного в поле формы. Обратите внимание на то, как вызывается метод `element()`.

```
...
if ($('form').validate().element($(e.target))) {
...
```

Эта цепочка вызовов запускает проверку измененного элемента, а возвращаемый методом результат представляет собой логическое (булево) значение, указывающее на корректность введенного значения. Тестирование всего выражения оператором `if` позволяет избежать прибавления некорректных значений к постоянно обновляющемуся счетчику суммарного количества выбранных единиц цветочной продукции.

Добавление дистанционной проверки

Виды проверки, которые выполняются в предыдущем примере и описывались в главе 13, служат примером так называемой *локальной проверки* (local validation). Этот термин относится к проверке, правила выполнения которой и данные, которые должны подчиняться этим правилам, доступны внутри документа.

Подключаемый модуль Validation поддерживает также *дистанционную проверку* (remote validation), при которой значения, введенные пользователем, пересылаются на сервер, где к ним применяются правила. Этой возможностью удобно пользоваться в тех случаях, когда отправка данных браузеру нежелательна ввиду их многочисленности, из соображений безопасности или просто потому, что вы хотите выполнить проверку с учетом самых последних данных.

Предупреждение. Дистанционная проверка требует соблюдения некоторых мер предосторожности, поскольку связанная с ней дополнительная нагрузка на сервер может оказаться значительной. В нашем примере дистанционная проверка выполняется всякий раз, когда пользователь изменяет значение в поле ввода, но в реальном приложении такой подход, вероятнее всего, привел бы к генерации слишком большого количества запросов. Более разумный подход состоит в том, чтобы выполнять дистанционную проверку лишь в качестве вспомогательного шага перед отправкой формы.

В главе 13 дистанционная проверка не описывалась, поскольку этот вид проверки связан с использованием данных в формате JSON и возможностей Ajax, и мне не хотелось преждевременно затрагивать эту тему. В листинге 16.7 показано, как организовать дистанционную проверку того, что пользователь не пытается заказать больше цветов, чем их имеется в наличии по данным сервера.

Листинг 16.7. Выполнение дистанционной проверки

```
...
$.when(flowerReq, jsonReq).then(function() {
    $('input').each(function(index, elem) {
        $(elem).rules("add", {
            required: true,
            min: 0,
            digits: true,
            remote: {
                url: "http://node.jacquisflowershop.com
                    /stockcheck",
                type: "post",
                global: false
            }
            messages: {
                required: "Пожалуйста, введите
                          количество " + plurals[elem.name],
                digits: "Пожалуйста, введите количество " +
                             plurals[elem.name],
                min: "Пожалуйста, введите положительное
                      число для " + plurals[elem.name]
                }
            })
        }).change(function(e) {
            if ($('form').validate()
                    .element($(e.target))) {
                var total = 0;
                $('input').each(function(index, elem) {
                    total += Number($(elem).val());
                });
                $('#total').text(total);
            }
    });
});
...
```

Видите, как легко организовать дистанционную проверку, располагая поддержкой Ajax в jQuery? В этом примере параметр `url` используется для указания URL-адреса ресурса, который будет вызван для выполнения дистанционной проверки, параметр `type` — для указания типа запроса (в данном случае POST), а параметр `global` — для отключения глобальных событий.

Я отключил глобальные события, поскольку не хотел, чтобы ошибки, которые инициируют запросы, связанные с проверкой, рассматривались наравне с обычными ошибками, вызванными другими действиями пользователя. Вместо этого я организую их обработку так, чтобы дальнейшую проверку выполнил сервер после отправки формы (сервер Node.js такую проверку не выполняет, но в реальных веб-приложениях это обязательно должно быть предусмотрено, о чем уже говорилось в главе 13).

Подключаемый модуль Validation использует ваши настройки Ajax для направления запроса по указанному URL-адресу, пересылая значение атрибута `name` элемента `input` и значение, введенное пользователем. Если сервер ответит словом `true`, то значение корректно. Любой другой ответ считается сообщением об ошибке, которое должно быть отображено для пользователя. Пример использования таких сообщений показан на рис. 16.3.

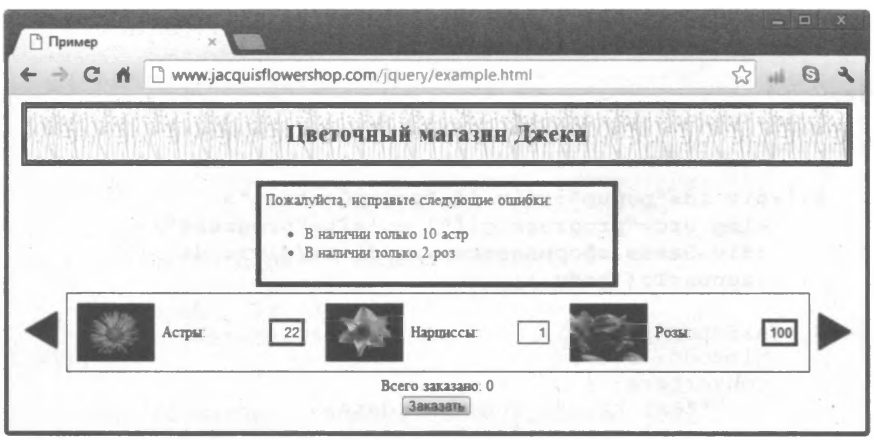


Рис. 16.3. Отображение сообщений дистанционной проверки

Отправка данных формы с использованием Ajax

Отправка значений формы — очень простой процесс, в примере реализации которого, приведенном в листинге 16.8, используется та же методика, что и в главе 15.

Листинг 16.8. Отправка формы с использованием Ajax

```
...
<style type="text/css">
    a.arrowButton {
        background-image: url(leftarrows.png); float: left;
        margin-top: 15px; display: block; width: 50px;
        height: 50px;}
    #right {background-image: url(rightarrows.png)}
    h1 { min-width: 0px; width: 95%; }
    #oblock { float: left; display: inline;
        border: thin black solid; }
    form { margin-left: auto; margin-right: auto;
        width: 885px; }
    #bbox {clear: left}
```

```
    #error {color: red; border: medium solid red;
        padding: 4px; margin: auto; width: 300px;
        text-align: center; margin-bottom: 5px}
    .invalidElem {border: medium solid red}
    #errorSummary {border: thick solid red; color: red;
        width: 350px; margin: auto; padding: 4px;
        margin-bottom: 5px}
    #popup {
        text-align: center; position: absolute; top: 100px;
        left: 0px; width: 100%; height: 1px;
        overflow: visible; visibility: visible;
        display: block }
    #popupContent { color: white; background-color: black;
        font-size: 14px ; font-weight: bold;
        margin-left: -75px; position: absolute; top: -55px;
        left: 50%; width: 150px; height: 60px;
        padding-top: 10px; z-index: 2;
    }
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('<div id="popup"><div id="popupContent">
            <img src="progress.gif"' + 'alt="progress"/>
            <div>Заказ оформляется...</div></div></div>')
            .appendTo('body');

        $.ajaxSetup({
            timeout: 5000,
            converters: {
                "text html": function(data)
                    { return $(data); }
            }
        })

        $(document).ajaxError(function(e, jqxhr,
                settings, errorMsg) {
            $('#error').remove();
            var msg = "Произошла ошибка. Пожалуйста,
                       повторите запрос"
            if (errorMsg == "timeout") {
                msg = "Время запроса истекло. Пожалуйста,
                       повторите запрос"
            } else if (jqxhr.status == 404) {
                    msg = "Файл не найден";
            }
            $('<div id=error/>').text(msg).insertAfter('h1');
        }).ajaxSuccess(function() {
            $('#error').remove();
        })

        $('#row2, #row3, #popup').hide();

        var flowerReq = $.get("flowers.html", function(data) {
            var elems = data.filter('div').addClass("dcell");
            elems.slice(0, 3).appendTo('#row1');
            elems.slice(3).appendTo("#row2");
```

```
})

var jsonReq = $.getJSON("additionalflowers.json",
    function(data) {
    $('#flowerTmpl').tmpl(data).appendTo("#row3");
})

var plurals = {
        astor: "астр", daffodil: "нарциссов", rose: "роз",
        peony: "пионов", primula: "примул",
        snowdrop: "подснежников", carnation: "гвоздик",
        lily: "лилий", orchid: "орхидей"
    }
$('<div id=errorSummary>Пожалуйста, исправьте
    следующие ошибки:</div>')
    .append('<ul id="errorsList"></ul>')
    .hide().insertAfter('h1');

$('form').validate({
    highlight: function(element, errorClass) {
        $(element).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).removeClass("invalidElem");
    },
    errorContainer: '#errorSummary',
    errorLabelContainer: '#errorsList',
    wrapper: 'li',
    errorElement: "div"
});

$.when(flowerReq, jsonReq).then(function() {
    $('input').each(function(index, elem) {
        $(elem).rules("add", {
            required: true,
            min: 0,
            digits: true,
            remote: {
                url: "http://node.jacquisflowershop.com
                    /stockcheck",
                type: "post",
                global: false
            },
            messages: {
                required: "Пожалуйста, введите
                  количество " + plurals[elem.name],
                digits: "Пожалуйста, введите количество
                  " + plurals[elem.name],
                min: "Пожалуйста, введите положительное
                  число для " + plurals[elem.name]
            }
        })
    }).change(function(e) {
        if ($('form').validate()
            .element($(e.target))) {
            var total = 0;
            $('input').each(function(index, elem) {
                total += Number($(elem).val());
```

```
            });
            $('#total').text(total);
        }
    });
});
$('button').click(function(e) {
    e.preventDefault();

    var formData = $('form').serialize();
    $('body *').not('#popup, #popup *')
        .css("opacity", 0.5);
    $('input').attr("disabled", "disabled");
    $('#popup').show();
    $.ajax({
        url:
            "http://node.jacquisflowershop.com:9999/
                order",
        type: "post",
        data: formData,
        complete: function() {
            setTimeout(function() {
            $('body *').not('#popup, #popup *')
                .css("opacity", 1);
            $('input').removeAttr("disabled");
            $('#popup').hide();
            }, 1500);
        }
    })
})
$('<a id=left></a><a id=right></a>').prependTo('form')
    .addClass("arrowButton").click(handleArrowPress)
    .hover(handleArrowMouse);
$('#right').appendTo('form');
var total = $('#buttonDiv')
    .prepend("<div>Всего заказано: <span id=total>0
        </span></div>")
    .css({clear: "both", padding: "5px"});
$('<div id=bbox />').appendTo("body").append(total);

function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ?
        "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}

function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];

    var visibleRow = $('div.drow:visible');
    var visibleRowIndex = jQuery
        .inArray(visibleRow.attr("id"),elemSequence);

    var targetRowIndex;

    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0)
            {targetRowIndex = elemSequence.length -1};
```

```
            } else {
                targetRowIndex = (visibleRowIndex + 1) %
                    elemSequence.length;
            }
            visibleRow.fadeOut("fast", function() {
                $('#' + elemSequence[targetRowIndex])
                .fadeIn("fast")});
        }
    });
</script>
...
```

Здесь я не ограничился одним лишь POST-запросом Ajax, поскольку мне хотелось предоставить некоторый дополнительный контекст, приближающий пример к ситуациям, в которых эти запросы могут обрабатываться в реальных проектах. Я начал с того, что добавил в документ новый элемент, извещающий пользователя о том, что его заказ принят, и расположил этот элемент над всеми остальными элементами. Реализующие это инструкции CSS и jQuery приведены ниже.

```
...
#popup {
    text-align: center; position: absolute; top: 100px;
    left: 0px; width: 100%; height: 1px;
    overflow: visible; visibility: visible;
    display: block }
#popupContent { color: white; background-color: black;
    font-size: 14px ; font-weight: bold;
    margin-left: -75px; position: absolute; top: -55px;
    left: 50%; width: 150px; height: 60px;
    padding-top: 10px; z-index: 2;
}
...
$('<div id="popup"><div id="popupContent">
    <img src="progress.gif"' + 'alt="progress"/>
    <div>Заказ оформляется</div></div></div>')
    .appendTo('body');
...
```

Создать элемент, который выглядел бы, как всплывающий, и занимал нужную позицию на экране, на удивление трудно, в чем можно убедиться, взглянув, сколько инструкций CSS для этого понадобилось. Сами же HTML-элементы очень просты, и сгенерированный HTML-фрагмент после некоторого дополнительного форматирования принимает следующий вид.

```
<div id="popup">
    <div id="popupContent">
        <img src="progress.gif" alt="progress">
        <div>Заказ оформляется</div>
    </div>
</div>
```

Указанный здесь элемент `img` (`progress.gif`) — это анимированное изображение в формате GIF. Существует ряд веб-сайтов, которые создают заказные изображения для индикаторов выполнения, и я воспользовался одним из них. Если вы не хотите заниматься этим сами, можете использовать изображение из архива примеров, доступного на сайте книги (см. главу 1). Вид этого индикатора на экране показан на рис. 16.4 (чтобы его можно было лучше разглядеть, другие элементы были удалены).

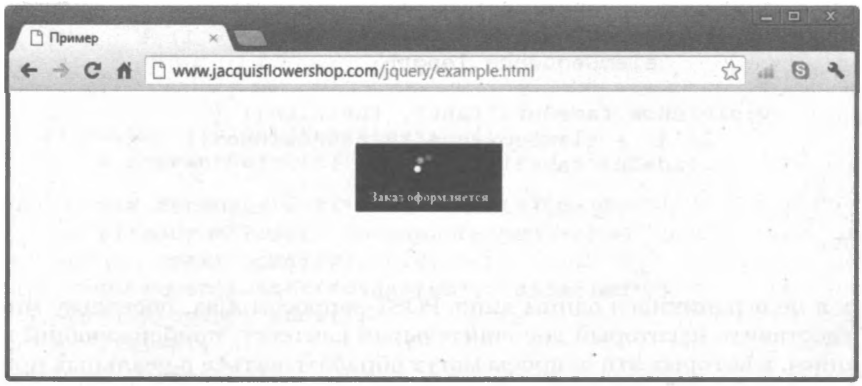


Рис. 16.4. Отображение индикатора хода выполнения

Первоначально я скрыл остальные элементы, поскольку нет смысла отображать индикатор для пользователя, если тот пока ничего не заказал.

```
$('#row2, #row3, #popup').hide();
```

Поместив эти элементы в нужное место и скрыв их, можно приступить к отправке формы. Мы регистрируем функцию-обработчик для события `click` элемента `button` следующим образом.

```
...
$('button').click(function(e) {
    e.preventDefault();

    var formData = $('form').serialize();
    $('body *').not('#popup, #popup *')
        .css("opacity", 0.5);
    $('input').attr("disabled", "disabled");
    $('#popup').show();
    $.ajax({
        url:
            "http://node.jacquisflowershop.com:9999/order",
        type: "post",
        data: formData,
        complete: function() {
            setTimeout(function() {
                $('body *').not('#popup, #popup *')
                    .css("opacity", 1);
                $('input').removeAttr("disabled");
                $('#popup').hide();
            }, 1500);
        }
    })
})
...
```

Прежде чем приступить к выполнению Ajax-запроса, мы отображаем все всплывающие элементы и делаем остальные элементы частично прозрачными. Кроме того, мы отключаем элементы `input`, добавляя к ним атрибут `disable`. Это делается для того, чтобы пользователь не мог изменить значение любого из этих элементов, пока ему пересылаются данные.

```
...
$('body *').not('#popup, #popup *')
    .css("opacity", 0.5);
$('input').attr("disabled", "disabled");
$('#popup').show();

...
```

Проблема с отключением функциональности элементов `input` состоит в том, что их значения не будут включаться в данные, отправляемые серверу. Дело в том, что метод `serialize()` будет включать в возвращаемый результат лишь значения, полученные из тех элементов `input`, которые, в соответствии с терминологией спецификации HTML 4.01, являются так называемыми *удачными элементами управления* (successful controls). В частности, из числа элементов формы, данные которых считаются пригодными для отправки серверу, исключаются элементы с отключенной функциональностью, а также элементы, не имеющие атрибута `name`. В принципе, можно было бы организовать просмотр элементов и самостоятельно получить все значения в любом случае, но более простой способ состоит в том, чтобы собрать необходимые для отправки данные еще до отключения функциональности соответствующих элементов.

```
...
complete: function() {
    setTimeout(function() {
        $('body *').not('#popup, #popup *')
            .css("opacity", 1);
        $('input').removeAttr("disabled");
        $('#popup').hide();
    }, 1500);
}
...
```

В реальном веб-приложении я этого не делал бы, но в демонстрационных целях, когда компьютер, на котором выполняется разработка, и сервер находятся в одной сети, полезно сделать этот переход более заметным. Окно браузера в процессе выполнения Ajax-запроса показано на рис. 16.5.

Рис. 16.5. Окно браузера в процессе запроса отправки формы

Обработка ответа от сервера

Нам осталось только сделать что-нибудь полезное с данными, получаемыми обратно от сервера. В этой главе мы используем простую таблицу. О том, как создавать функционально насыщенные пользовательские интерфейсы с помощью библиотеки jQuery UI, рассказывается в следующей части, и я не хочу делать вручную то, что можно сделать гораздо элегантнее с помощью виджетов. Окончательный результат представлен на рис. 16.6.

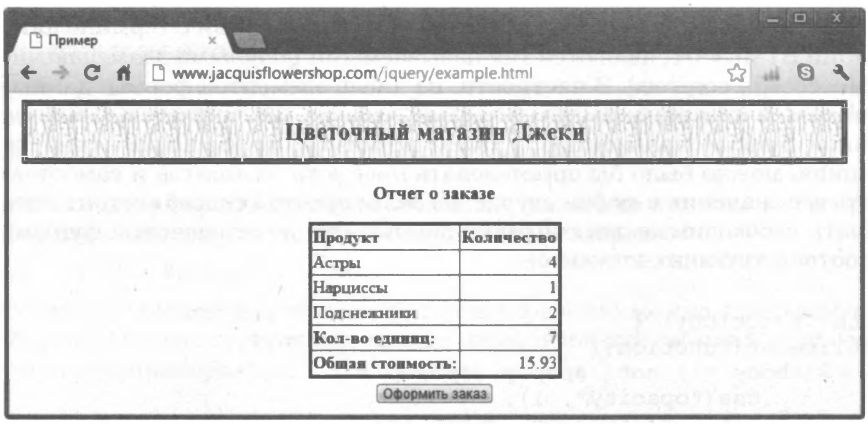


Рис. 16.6. Отображение отчета о заказе

Полный текст документа, который поддерживает это улучшение, приведен в листинге 16.9.

Листинг 16.9. Обработка ответа, полученного от сервера

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery.validate.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        a.arrowButton {
            background-image: url(leftarrows.png); float: left;
            margin-top: 15px; display: block; width: 50px;
            height: 50px;}
        #right {background-image: url(rightarrows.png)}
        h1 { min-width: 0px; width: 95%; }
        #oblock { float: left; display: inline;
            border: thin black solid; }
        #orderForm { margin-left: auto; margin-right: auto;
            width: 885px; }
        #bbox {clear: left}
        #error {color: red; border: medium solid red;
            padding: 4px; margin: auto; width: 300px;
```

```
        text-align: center; margin-bottom: 5px}
    .invalidElem {border: medium solid red}
    #errorSummary {border: thick solid red; color: red;
        width: 350px; margin: auto; padding: 4px;
        margin-bottom: 5px}
    #popup {
        text-align: center; position: absolute; top: 100px;
        left: 0px; width: 100%; height: 1px;
        overflow: visible; visibility: visible;
        display: block }
    #popupContent { color: white; background-color: black;
        font-size: 14px ; font-weight: bold;
        margin-left: -75px; position: absolute; top: -55px;
        left: 50%; width: 150px; height: 60px;
        padding-top: 10px; z-index: 2;
    }
    #summary {text-align: center}
    table {border-collapse: collapse;
        border: medium solid black; font-size: 18px;
        margin: auto; margin-bottom: 5px;}
    th {text-align: left}
    th, td {padding: 2px}
    tr > td:nth-child(1) {text-align: left}
    tr > td:nth-child(2) {text-align: right}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('<div id="popup"><div id="popupContent">
            <img src="progress.gif"' +
            + 'alt="выполняется..."/>
            <div>Оформляется заказ...</div></div></div>')
            .appendTo('body');

        $.ajaxSetup({
            timeout: 5000,
            converters: {
               "text html": function(data) { return $(data); }
            }
        })

        $(document)
           .ajaxError(function(e, jqxhr, settings, errorMsg) {
            $('#error').remove();
            var msg = "Произошла ошибка. Пожалуйста,
                повторите запрос"
            if (errorMsg == "timeout") {
                msg = "Время запроса истекло. Пожалуйста,
                повторите запрос"
            } else if (jqxhr.status == 404) {
                    msg = "Файл не найден";
            }
            $('<div id=error/>').text(msg).insertAfter('h1');
        }).ajaxSuccess(function() {
            $('#error').remove();
        })
```

```
$('#row2, #row3, #popup, #summaryForm').hide();

var flowerReq = $.get("flowers.html", function(data) {
    var elems = data.filter('div').addClass("dcell");
    elems.slice(0, 3).appendTo('#row1');
    elems.slice(3).appendTo("#row2");
})

var jsonReq = $
   .getJSON("additionalflowers.json", function(data) {
    $('#flowerTmpl').tmpl(data).appendTo("#row3");
})

var plurals = {
    astor: "астр", daffodil: "нарциссов", rose: "роз",
    peony: "пионов", primula: "примул",
    snowdrop: "подснежников", carnation: "гвоздик",
    lily: "лилий", orchid: "орхидей"
}

$('<div id=errorSummary>Пожалуйста, исправьте
    следующие ошибки:</div>')
    .append('<ul id="errorsList"></ul>')
    .hide().insertAfter('h1');

$('#orderForm').validate({
    highlight: function(element, errorClass) {
        $(element).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).removeClass("invalidElem");
    },
    errorContainer: '#errorSummary',
    errorLabelContainer: '#errorsList',
    wrapper: 'li',
    errorElement: "div"
});

$.when(flowerReq, jsonReq).then(function() {
    $('input').each(function(index, elem) {
        $(elem).rules("add", {
            required: true,
            min: 0,
            digits: true,
            remote: {
                url: "http://node.jacquisflowershop
                    .com:9999/stockcheck",
                type: "post",
                global: false
            },
            messages: {
              required: "Пожалуйста, введите
                  количество " + plurals[elem.name],
              digits: "Пожалуйста, введите количество
                  " + plurals[elem.name],
              min: "Пожалуйста, введите положительное
                  число для " + plurals[elem.name]
            }
```

```
        })
    }).change(function(e) {
        if ($('#orderForm').validate()
            .element($(e.target))) {
            var total = 0;
            $('input').each(function(index, elem) {
                total += Number($(elem).val());
            });
            $('#total').text(total);
        }
    });
});

$('#orderForm button').click(function(e) {
    e.preventDefault();

    var formData = $('#orderForm').serialize();
    $('body *').not('#popup, #popup *')
        .css("opacity", 0.5);
    $('input').attr("disabled", "disabled");
    $('#popup').show();
    $.ajax({
        url: "http://node.jacquisflowershop
            .com:9999/order",
        type: "post",
        data: formData,
        dataType: "json",
        dataFilter: function(data, dataType) {
            data = $.parseJSON(data);

            var cleanData = {
                totalItems: data.totalItems,
                totalPrice: data.totalPrice
            };
            delete data.totalPrice;
            delete data.totalItems;
            cleanData.products = [];
            for (prop in data) {
                cleanData.products.push({
                    name: plurals[prop],
                    quantity: data[prop]
                })
            }
            return cleanData;
        },
        converters: {"text json": function(data)
            { return data;}},
        success: function(data) {
            processServerResponse(data);
        },
        complete: function() {
            $('body *').not('#popup, #popup *')
                .css("opacity", 1);
            $('input').removeAttr("disabled");
            $('#popup').hide();
        }
```

```
    })
})

function processServerResponse(data) {
    if (data.products.length > 0) {
        $('body > *:not(h1)').hide();
        $('#summaryForm').show();
        $('#productRowTmpl').tmpl(data.products)
            .appendTo('tbody');
        $('#totalItems').text(data.totalItems);
        $('#totalPrice').text(data.totalPrice);
    } else {
        var elem = $('input').get(0);
            var err = new Object();
            err[elem.name] = "Не выбран ни один
                продукт";
        $('#orderForm').validate().showErrors(err);
        $(elem).removeClass("invalidElem");
    }
}

$('<a id=left></a><a id=right></a>')
    .prependTo('#orderForm')
    .addClass("arrowButton")
    .click(handleArrowPress).hover(handleArrowMouse);
$('#right').appendTo('#orderForm');

var total = $('#buttonDiv')
    .prepend("<div>Всего заказано:
         <span id=total>0</span></div>")
    .css({clear: "both", padding: "5px"});
$('<div id=bbox />').appendTo("body").append(total);

function handleArrowMouse(e) {
   var propValue = e.type == "mouseenter" ?
        "-50px 0px" : "0px 0px";
   $(this).css("background-position", propValue);
}

function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];
    var visibleRow = $('div.drow:visible');
    var visibleRowIndex = jQuery
        .inArray(visibleRow.attr("id"),elemSequence);
    var targetRowIndex;
    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0)
            {targetRowIndex = elemSequence.length -1};
    } else {
        targetRowIndex = (visibleRowIndex + 1) %
            elemSequence.length;
    }
    visibleRow.fadeOut("fast", function() {
        $('#' + elemSequence[targetRowIndex])
            .fadeIn("fast")});
}
```

```
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
      <script id="productRowTmpl" type="text/x-jquery-tmpl">
        <tr><td>${name}</td><td>${quantity}</td></tr>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form id="orderForm" method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
                <div id="row3" class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv"><button
type="submit">Заказать</button></div>
    </form>
    <form id="summaryForm" method="post" action="">
        <div id="summary">
            <h3>Отчет о заказе</h3>
            <table border="1">
                <thead>
                    <tr><th>Продукт</th><th>Количество</th>
                </thead>
                <tbody>
                </tbody>
                <tfoot>
                    <tr><th>Кол-во единиц:</th>
                    <td id="totalitems"></td></tr>
                    <tr><th>Общая стоимость:</th>
                    <td id="totalprice"></td></tr>
                </tfoot>
            </table>
            <div id="buttonDiv2">
                <button type="submit">Оформить заказ</button>
            </div>
        </div>
    </form>
</body>
</html>
```

Добавление новой формы

Первое, что мы сделали, — это добавили новую форму в статическую часть HTML-кода документа.

```
...
<form id="summaryForm" method="post" action="">
    <div id="summary">
        <h3>Отчет о заказе</h3>
        <table border="1">
            <thead>
                <tr><th>Продукт</th><th>Количество</th>
            </thead>
            <tbody>
            </tbody>
            <tfoot>
                <tr><th>Кол-во единиц:</th>
                    <td id="totalitems"></td></tr>
                <tr><th>Общая стоимость:</th>
                    <td id="totalprice"></td></tr>
            </tfoot>
        </table>
        <div id="buttonDiv2"><button type="submit">
            Оформить заказ</button></div>
    </div>
</form>
...
```

Эта форма образует ядро новой функциональности. Когда пользователь отправит список выбранных им продуктов на сервер, таблица в этой форме будет использоваться для отображения данных, полученных обратно в результате выполнения Ajax-запроса.

Совет. В предыдущих примерах использовался селектор `$('form')`, но поскольку теперь в документе содержатся две формы, то соответствующие ссылки были изменены таким образом, чтобы использовались атрибуты `id` элементов `form`.

Мы не хотим сразу же отображать эту форму, и поэтому включаем ее в список скрытых элементов.

```
$('#row2, #row3, #popup, #summaryForm').hide();
```

Как вы теперь, наверное, уже и сами догадываетесь, появление новых элементов всегда сопровождается созданием для них новых CSS-стилей.

```
...
#summary {text-align: center}
table {border-collapse: collapse; border: medium solid black;
    font-size: 18px; margin: auto; margin-bottom: 5px;}
th {text-align: left}
th, td {padding: 2px}
tr > td:nth-child(1) {text-align: left}
tr > td:nth-child(2) {text-align: right}
...
```

Эти стили гарантируют, что таблица будет отображаться посередине окна браузера, а текст в различных колонках будет выровнен правильно.

Выполнение Ajax-запроса

Следующий шаг заключается в вызове метода `ajax()` для выполнения Ajax-запроса.

```
...
$('#orderForm button').click(function(e) {
    e.preventDefault();
```

```
    var formData = $('#orderForm').serialize();
    $('body *').not('#popup, #popup *').css("opacity", 0.5);
    $('input').attr("disabled", "disabled");
    $('#popup').show();
    $.ajax({
        url: "http://node.jacquisflowershop.com:9999/order",
        type: "post",
        data: formData,
        dataType: "json",
        dataFilter: function(data, dataType) {
            data = $.parseJSON(data);

            var cleanData = {
                totalItems: data.totalItems,
                totalPrice: data.totalPrice
            };
            delete data.totalPrice; delete data.totalItems;
            cleanData.products = [];
            for (prop in data) {
                cleanData.products.push({
                    name: plurals[prop],
                    quantity: data[prop]
                })
            }
            return cleanData;
        },
        converters: {"text json": function(data) { return data;}},
        success: function(data) {
            processServerResponse(data);
        },
        complete: function() {
            $('body *').not('#popup, #popup *').css("opacity", 1);
            $('input').removeAttr("disabled");
            $('#popup').hide();
        }
    })
})
...
```

Здесь мы удалили явно заданную задержку в функции `complete` и добавили в запрос настройки `dataFilter`, `converters` и `success`.

Параметр `dataFilter` используется для предоставления функции, которая преобразует данные JSDON, полученные с сервера, в нечто более полезное. Ответом сервера является примерно такая строка в формате JSON.

```
{"astor":"4","daffodil":"1","snowdrop":"2",
 "totalItems":7,"totalPrice":"15.93"}
```

В результате разбора JSON-данных и их реструктуризации мы получаем следующее.

```
{"totalItems":7,
 "totalPrice":"15.93",
 "products":[{"name":"Astors","quantity":"4"},
             {"name":"Daffodils","quantity":"1"},
             {"name":"Snowdrops","quantity":"2"}]
}
```

Этот формат обладает двумя преимуществами. Во-первых, он больше приспособлен для использования с шаблонами данных, поскольку позволяет передавать свойство `products` методу `tmpl`. Во-вторых, он позволяет проверить с помощью свойства `products.length`, выбрал ли вообще пользователь какой-либо продукт. В целом указанные два преимущества не очень значительны, но мне хотелось интегрировать в этот пример как можно больше возможностей, описанных в предыдущих главах. Также обратите внимание на замену имени продукта (например, `orchid`) его множественным числом (например, `Orchids`).

Преобразовав данные JSON в объект JavaScript (с помощью метода `parseJSON()`, который описывается в главе 33), мы хотим отключить встроенный конвертер данных, который будет пытаться сделать то же самое. Для этого мы определяем пользовательский конвертер для данных JSON, который пропускает данные, не изменяя их.

```
...
converters: {"text json": function(data) { return data;}}
...
```

Обработка данных

В вызове метода `ajax()` мы указали в параметре `success` функцию `processServerResponse()`, которую определили следующим образом.

```
...
function processServerResponse(data) {
    if (data.products.length > 0) {
        $('body > *:not(h1)').hide();
        $('#summaryForm').show();
        $('#productRowTmpl').tmpl(data.products)
            .appendTo('tbody');
        $('#totalitems').text(data.totalItems);
        $('#totalprice').text(data.totalPrice);
    } else {
        var elem = $('input').get(0);
            var err = new Object();
            err[elem.name] = "Не выбран ни один продукт";
        $('#orderForm').validate().showErrors(err);
        $(elem).removeClass("invalidElem");
    }
}
...
```

Если в полученных от сервера данных содержится информация о продуктах, мы скрываем в документе все нежелательные элементы (включая элемент `form` и все дополнительные элементы, добавленные в сценарии) и отображаем новую форму. Таблица заполняется с использованием следующего шаблона данных.

```
<script id="productRowTmpl" type="text/x-jquery-tmpl">
  <tr><td>${name}</td><td>${quantity}</td></tr>
</script>
```

Это очень простой шаблон, который генерирует ряд таблицы для каждого выбранного пользователем продукта. Наконец, мы задаем содержимое ячеек таблицы, которое отображает количество заказанных наименований и общую стоимость товара с помощью метода `text()`.

```
$('#totalitems').text(data.totalItems);
$('#totalprice').text(data.totalPrice);
```

В то же время, если в полученных от сервера данных отсутствует какая-либо информация о продуктах (это говорит о том, что пользователь оставил во всех полях ввода нулевые значения), то мы действуем иначе. Сначала мы выбираем первый из элементов `input`.

```
var elem = $('input').get(0);
```

Затем мы создаем объект, содержащий свойство, именем которого является значение атрибута `name` элемента `input`, а значением — сообщение, выводимое для пользователя. После этого для элемента `form` вызывается метод `validate()`, а для элемента `result` — метод `showErrors()`.

```
var err = new Object();
err[elem.name] = "Не выбран ни один продукт";
$('#orderForm').validate().showErrors(err);
```

Это позволяет вручную вводить сообщения об ошибках в систему проверки и пользоваться всеми преимуществами их структуризации и форматирования, которые мы ранее предусмотрели в шаблоне. Чтобы подключаемый модуль Validation мог визуально выделять элемент, с которым связана ошибка, мы должны предоставить имя элемента, что является не совсем идеальным решением, как показано на рис. 16.7.

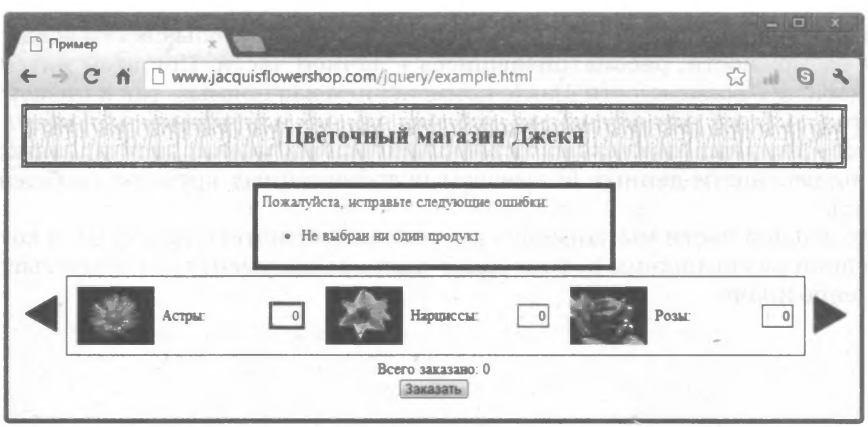


Рис. 16.7. Отображение сообщения об ошибке при выборе продукции

Недостатком является то, что отображаемое сообщение носит общий характер, но визуальное выделение применяется лишь к одному элементу `input`. Чтобы справиться с этим, мы удаляем класс, который используется модулем проверки для визуального выделения.

```
$(elem).removeClass("invalidElem");
```

Это приводит к результату, представленному на рис. 16.8.

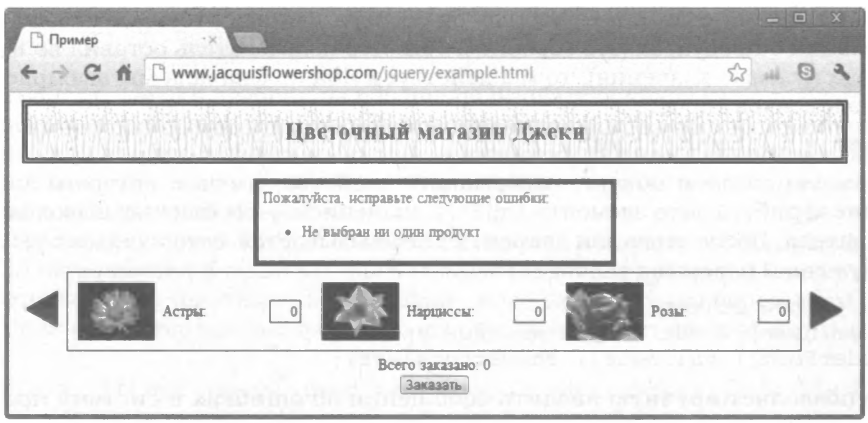


Рис. 16.8. Удаление визуального выделения элемента, с которым связана ошибка

Резюме

В этой главе мы выполнили рефакторинг примера, чтобы свести воедино все темы и возможности, рассматривавшиеся в данной части. При этом интенсивно использовались возможности Ajax (с применением как прямых, так и низкоуровневых методов), были применены два шаблона данных и подключаемый модуль Validation, обеспечивший возможность выполнения локальной и дистанционной проверки корректности данных (с выводом подготовленных вручную сообщений об ошибках).

В следующей части мы займемся изучением библиотеки jQuery UI, и когда мы в очередной раз выполним рефакторинг примера документа, он будет выглядеть совершенно иначе.

Часть IV

Использование библиотеки jQuery UI

ГЛАВА 17

Установка библиотеки jQuery UI

Загрузить и установить библиотеку jQuery UI в некотором смысле сложнее, чем другие библиотеки JavaScript. Этот процесс не то чтобы трудоемок, но нуждается в пояснениях, чему и посвящена данная глава. Для работы с книгой вам достаточно будет установить версию библиотеки, предназначенную для разработки, но ниже также описывается, как установить минимизированные файлы, ориентированные на развертывание в производственной среде, и как использовать jQuery UI через сети распространения содержимого (CDN).

Получение библиотеки jQuery UI

Процесс загрузки библиотеки jQuery UI отличается несколько большей сложностью по сравнению с другими библиотеками JavaScript, но результат будет стоить затраченных усилий. Библиотека jQuery UI охватывает пять областей функциональности, и вам предоставляется возможность самостоятельно сконфигурировать загрузочный архив, включив в него лишь необходимые компоненты. В этой части вы познакомитесь со всеми возможностями библиотеки jQuery UI, но при работе с реальными веб-приложениями можно исключать ненужные компоненты для уменьшения размера библиотеки, загружаемой браузерами.

Совет. Библиотека jQuery UI — не единственный набор инструментальных средств для разработки пользовательских интерфейсов (UI) на базе jQuery. Например, имеется библиотека с открытым исходным кодом jQuery Tools, доступная для загрузки без каких-либо лицензий и ограничений на сайте `http://flowplayer.org/tools`. Также существует ряд коммерческих библиотек, таких как jQWidgets (`www.jqwidgets.com`) или Wijmo (`http://wijmo.com`). И конечно же, имеется библиотека jQuery Mobile, которой посвящена часть IV.

Выбор темы оформления

Прежде чем приступить к созданию собственной библиотеки jQuery UI, вы должны выбрать тему оформления. Библиотека jQuery UI предлагает богатейшие возможности и способы выбора конфигураций, благодаря чему можно с легкостью изменить внешний вид любого используемого средства. В действительности число доступных возможностей выбора настолько велико, что иногда это поистине ошеломляет. На сайте jQuery UI можно воспользоваться услугами специального приложения — настройщика тем (Themeroller), но кроме того существует целая галерея предопреде-

ленных тем, полностью готовых к использованию, из которых можно выбрать ту, которая вас больше всего устраивает, и тем самым облегчить себе жизнь.

Начните с посещения сайта `jqueryui.com` и щелкните на кнопке Themes. В результате откроется страница ThemeRoller, отображающая виджеты jQuery UI и расположенную слева от них панель настроек, с помощью которой можно установить параметры темы оформления, как показано на рис. 17.1.

Рис. 17.1. Страница выбора тем оформления на сайте jQuery UI

Если у вас уже используется определенный визуальный стиль, которого вы должны придерживаться, и вы хотите, чтобы визуальный интерфейс средств jQuery UI согласовывался с остальной частью сайта или приложения, то вкладка Roll Your Own (которая выбирается по умолчанию) — это как раз то, что нужно. Можно изменить любой аспект оформления с помощью набора стилей CSS, который используется библиотекой jQuery UI.

Чтобы получить одну из готовых тем, следует перейти на вкладку Gallery. На момент написания этих строк галерея включала 24 темы, охватывающие широкий спектр вариантов цветового оформления — от приглушенных и нежных тонов до ярких и вызывающих. При выполнении щелчков на темах галереи внешний вид виджетов, отображаемых на остальной части страницы, будет соответствующим образом обновляться, позволяя вам оценить, как может выглядеть приложение (рис. 17.2).

Используемая для jQuery UI стандартная тема носит название *UI lightness*, но эта тема недостаточно контрастна для воспроизведения используемой в ней цветовой схемы на книжных страницах, и поэтому я буду использовать тему *Sunny*, которая выглядит немного лучше. Единственное, что от вас сейчас требуется, — это запомнить название темы, которая вас устраивает. В полиграфическом исполнении темы выглядят не особенно привлекательно, но, будучи отображенными на

экране, они производят совершенно иное впечатление. Я рекомендую просмотреть весь перечень тем и выбрать ту из них, которая придется вам по душе.

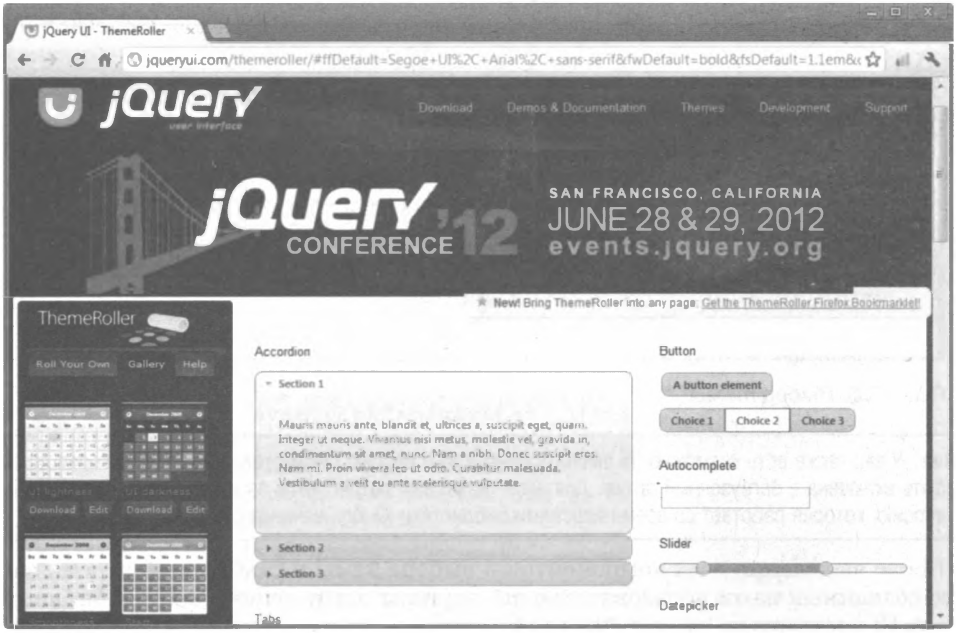


Рис. 17.2. Демо-галерея, в которой отображается тема оформления Sunny

Совет. Вы вовсе не обязаны выбирать ту же тему, что и я, но если ваш выбор будет другим, то и получаемые вами результаты, несомненно, будут выглядеть не так, как мои.

Создание настраиваемого загрузочного архива библиотеки jQuery UI

Выбрав для себя определенную тему оформления, можете приступить к созданию собственного варианта загрузки библиотеки jQuery UI. Щелкните на кнопке Download в верхней части страницы для перехода на страницу Build Your Own Download. Вы увидите список компонентов jQuery UI, разбитых на четыре функциональные группы: UI Core, Interactions, Widgets и Effects.

Выбирая лишь те возможности, которые действительно нужны вашему проекту, вы уменьшите размер набора файлов, который должны будут загружать браузеры. Сама по себе эта идея неплохая, но я придерживаюсь другого подхода. С моей точки зрения, гораздо лучше сэкономить часть полосы пропускания своего канала связи и переложить задачу доставки jQuery UI в браузеры на одну из сетей распространения содержимого, о чем будет говориться далее.

Для данной главы вам понадобятся все компоненты, поэтому проследите за тем, чтобы были установлены все флажки.

Следующий шаг заключается в выборе темы. Это делается с помощью раскрывающегося списка, который располагается справа на странице, как показано на рис. 17.3.

Совет. Между некоторыми компонентами, фигурирующими в списке, существуют зависимости, но в процессе создания своего варианта библиотеки можете об этом не думать. Если вы выбираете какой-либо компонент, то одновременно с ним загрузятся все компоненты, от которых он зависит.

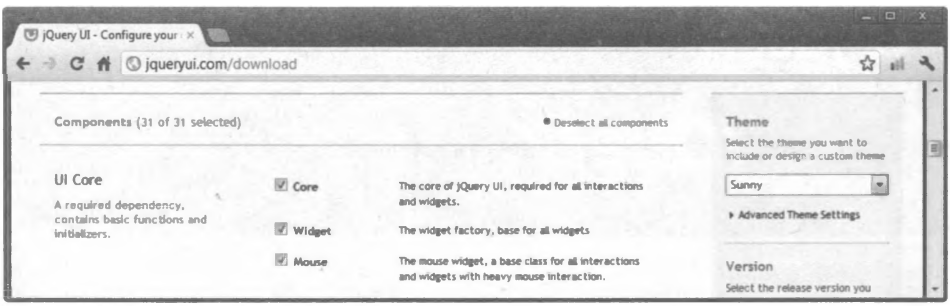


Рис. 17.3. Выбор темы

Совет. У вас также есть возможность выбрать конкретную версию библиотеки jQuery UI, которая должна быть включена в загрузочный архив. Для этой главы вам потребуется загрузить *стабильную* (Stable) версию, которая работает со всеми версиями библиотеки jQuery, начиная с версии 1.3.2.

После выделения всех компонентов и выбора темы и стабильной версии загрузите созданный вами пользовательский вариант загрузочного архива библиотеки jQuery UI, щелкнув на кнопке Download.

Установка версии библиотеки jQuery UI, предназначенной для разработки

Загрузочный архив jQuery UI содержит все файлы, необходимые для использования библиотеки как в процессе разработки, так и в производственной среде. Для работы с примерами книги вам понадобятся файлы, которые содержат несжатый исходный код и предназначены для использования в процессе разработки. В случае возникновении каких-либо проблем вы сможете легко изучить код для ознакомления с внутренним устройством библиотеки jQuery UI, что окажет неоценимую помощь при отладке сценариев. Вы должны скопировать в папку с файлами примера следующие файлы и папки:

- `development-bundle\ui\jquery-ui-1.8.16.custom.js`;
- `development-bundle\themes\sunny\jquery-ui-1.8.16.custom.css`;
- папка `development-bundle\themes\sunny\images`.

Содержащиеся в папках `ui` и `themes` файлы JavaScript и CSS используются отдельными компонентами и средствами, входящих в состав библиотеки. У вас не будет необходимости обращаться к ним, но они могут пригодиться в том случае, если вы захотите работать с ограниченным набором средств библиотеки jQuery UI.

Совет. Имена JavaScript- и CSS-файлов включают номер версии загруженного выпуска библиотеки. В моем случае это версия 1.8.16. Библиотека jQuery UI активно разрабатывается, и вы можете загрузить более позднюю версию, чем 1.8.16.

Подключение библиотеки jQuery UI к HTML-документу

Все, что вам теперь остается сделать, — это включить библиотеку jQuery UI в свой HTML-документ. Это можно сделать, добавив в документ элементы `script` и `link`, содержащие ссылки на файлы JavaScript и CSS, которые вы загрузили, как показано в листинге 17.1. Ссылаться непосредственно на папку `images` необязательно. Коль скоро папка `images` и CSS-файл находятся на своих местах, jQuery UI сможет самостоятельно найти все необходимые ресурсы.

Листинг 17.1. Включение библиотеки jQuery UI в документ

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('a').button();
        });
    </script>
</head>
<body>
    <a href="http://apress.com">Посетите веб-сайт Apress</a>
</body>
</html>
```

Совет. Библиотека jQuery UI зависит от библиотеки jQuery. Для того чтобы использовать jQuery UI в документе, ее следует предварительно подключить к нему. Библиотека jQuery UI не относится к числу автономно используемых библиотек.

Показанный в листинге документ содержит простой тест, позволяющий проверить правильность подключения библиотеки jQuery UI. В случае нормального открытия страницы вы должны увидеть кнопку, похожую на ту, которая показана на рис. 17.4. Не обращайте пока внимания на вызов метода `button()` в элементе `script`. О том, для чего он предназначен и как работает, вы узнаете в главе 18.

В случае неправильного указания пути к любой из двух библиотек вы увидите вместо этого простую ссылку, как показано на рис. 17.5.

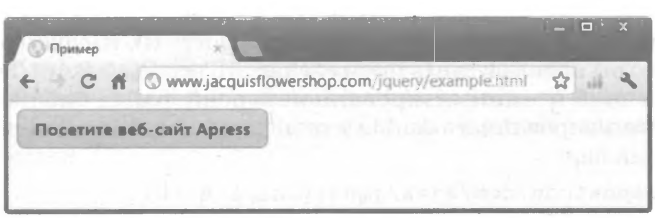


Рис. 17.4. Проверка корректности подключения библиотеки jQuery UI к документу

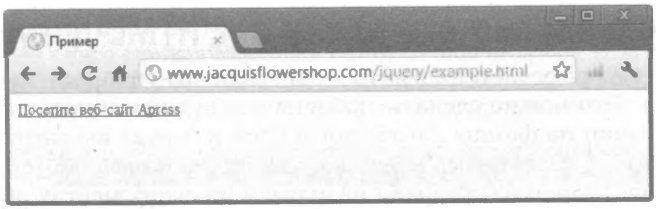


Рис. 17.5. Вид документа, в который не удалось импортировать библиотеку jQuery UI

Установка библиотеки jQuery UI для производственной среды

Завершив разработку своего веб-приложения и подготовившись к его развертыванию, можете использовать минимизированные версии файлов, включенных в загрузочный архив. Эти файлы имеют меньшие размеры, однако прочитать их содержимое, если это потребуется в интересах отладки, будет нелегко. Чтобы использовать версии файлов, предназначенные для развертывания, скопируйте в каталог своего веб-сервера следующие файлы и папки:

- `js\jquery-ui-1.8.16.custom.min.js`;
- `css\sunny\jquery-ui-1.8.16.custom.css`;
- папка `css\sunny\images`

Папка `images` и файл CSS здесь те же, что и в версии для разработки; изменения могут касаться лишь файла JavaScript. Чтобы выполнить чистую установку, достаточно скопировать эти файлы в каталог сервера.

Использование библиотеки jQuery UI через сеть распространения содержимого

Вопрос об использовании CDN для загрузки библиотеки jQuery уже затрагивался в главе 5. Если вы приверженец такого подхода, то вас порадует тот факт, что точно так же можно поступить и в случае библиотеки jQuery UI. Как Google, так и Microsoft обеспечивают хостинг файлов jQuery UI в своих сетях CDN. Для нашего базового примера я использую службу Microsoft, поскольку она предоставляет как JavaScript-файлы jQuery UI, так и стандартные темы оформления.

Чтобы использовать CDN, необходимо располагать URL-адресами нужных файлов. Если речь идет о службе Microsoft, то введите в браузере адрес `http://www.asp.net/ajaxlibrary/cdn.ashx`. Прокрутив страницу вниз, вы увидите список ссылок, соответствующих различным версиям jQuery UI. Щелкните на ссылке той версии, которую вы используете (в моем случае это версия 1.8.16). Вы увидите URL-адреса для обычной и минимизированной версий файла библиотеки jQuery UI. URL-адрес минимизированного файла для используемой мною версии библиотеки имеет следующий вид:

```
http://ajax.aspnetcdn.com/ajax/jquery.ui/1.8.16/
    jquery-ui.min.js
```

На оставшейся части страницы отображаются готовые темы, под каждой из которых указывается URL-адрес файла CSS. Теме *Sunny* соответствует следующий URL-адрес:

```
http://ajax.aspnetcdn.com/ajax/jquery.ui/1.8.16/themes/
sunny/jquery-ui.css
```

Чтобы подключить эти файлы к документу через CDN, достаточно поместить в элементы `script` и `link` не ссылки на локальные файлы jQuery UI, а соответствующие URL-адреса, как показано в листинге 17.2.

Листинг 17.2. Использование jQuery UI через CDN

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.8.16/
        jquery-ui.min.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.8.16/
      themes/sunny/jquery-ui.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('a').button();
        });
    </script>
</head>
<body>
    <a href="http://apress.com">Посетите веб-сайт Apress</a>
</body>
</html>
```

Опять-таки, признаком правильности указания URL-адресов будет служить отображение на открывшейся странице кнопки, аналогичной той, которая показана на рис. 17.4.

Резюме

В этой главе вы узнали о том, как создать загрузочный архив библиотеки jQuery UI. Вам предоставляется большая гибкость в выборе включаемых в библиотеку средств и настройке внешнего вида приложения, который будет по умолчанию обеспечивать jQuery UI. Мне особенно нравится приложение ThemeRoller. Оно предлагает изящный способ создания настраиваемой темы оформления, вписывающейся в уже существующую визуальную схему, что идеально подходит для использования библиотеки jQuery UI на корпоративных сайтах.

В следующей главе мы приступаем к рассмотрению возможностей jQuery UI, начав с обсуждения самого популярного функционального компонента этой библиотеки — виджетов.

ГЛАВА 18

Использование виджетов Button, Progress Bar и Slider

Теперь, когда вы сконфигурировали, загрузили и установили библиотеку jQuery UI, можем приступить к ознакомлению с содержащимися в ней виджетами. Виджеты — это главные функциональные блоки jQuery UI, и хотя эта библиотека обеспечивает множество других возможностей (например, эффекты, о которых речь пойдет в главе 34), именно виджеты принесли ей известность.

В данной главе описаны три простейших виджета: Button (кнопка), Progress Bar (индикатор процесса) и Slider (ползунок). Некоторые характеристики, в том числе свойства, методы и события, являются общими для всех виджетов. Освоив один виджет, вы создадите надежный фундамент для работы со всеми остальными виджетами, поэтому начало главы посвящено изучению некоего общего для всех виджетов контекста.

Связать все виджеты с нашим примером сайта цветочного магазина нелегко, поэтому пусть вас не удивляет, что многие примеры этой части будут представлять собой очень небольшие, независимые HTML-документы, демонстрирующие использование какого-то одного виджета. К примеру с цветочным магазином мы вернемся в главе 25, в которой он будет переработан с целью включения в него возможностей jQuery UI. Перечень тем, рассматриваемых в данной главе, приведен в табл. 18.1.

Таблица 18.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Создание виджета Button (кнопки jQuery UI)	Выберите элемент и используйте метод `button()`	1
Конфигурирование элемента `button`	Передайте объект отображения методу `button()` или используйте метод `option`	2, 3
Использование значков в кнопках jQuery UI	Используйте опцию `icons`	4
Применение пользовательских изображений в кнопках jQuery UI	Установите элемент `img` в качестве содержимого кнопки	5
Удаление кнопки jQuery UI	Используйте метод `destroy`	6

Окончание табл. 18.1

Задача	Решение	Листинг
Включение и отключение функциональности кнопки jQuery UI	Используйте метод `enable` или `disable`	7
Обновление состояния кнопки jQuery UI для отражения изменений базового элемента, выполненных программным путем	Используйте метод `refresh`	8
Реагирование на создание кнопки jQuery UI	Укажите функцию для события `create`	9
Создание однотипных кнопок из различных видов элементов	Создайте кнопки jQuery UI из элементов `input`, `button` и `a`	10
Создание кнопки-переключателя jQuery UI	Создайте кнопку jQuery UI на основе элемента `checkbox`	11
Создание наборов кнопок jQuery UI	Используйте метод `buttonset()`	12, 13
Создание виджета Progress Bar (индикатора процесса jQuery UI)	Используйте метод `progressbar()`	14
Получение и изменение значений индикатора процесса, отображаемых для пользователя	Используйте метод `value`	15
Анимация индикатора процесса jQuery UI	Используйте анимационное GIF-изображение в качестве значения свойства `background-image` в CSS-классе `ui-progressbar-value`	16
Реагирование на изменение состояния индикатора процесса	Укажите функции для событий `create`, `change` и `complete`	17
Создание виджета Slider (ползунка jQuery UI)	Используйте метод `slider()`	18
Изменение ориентации ползунка jQuery UI	Используйте опцию `orientation`	19, 20
Анимация ползунка jQuery UI при выполнении щелчка на нем	Используйте опцию `animate`	21
Создание ползунка jQuery UI, позволяющего указывать значения в некотором диапазоне	Используйте методы `range` и `values`	22
Управление ползунком jQuery UI из программы	Используйте методы `value` и `values`	23
Реагирование на изменение положения рукоятки ползунка	Обработайте события `start`, `stop`, `change` и `slide`	24

Использование виджета Button

Первый из виджетов, с которым вы начнете работать, предоставляет неплохую возможность познакомиться с миром jQuery UI. Виджет Button относительно прост, но это не мешает ему оказывать трансформирующее воздействие на HTML-документы. Данный виджет обеспечивает применение темы jQuery UI к элементам `button` и `a`. Это означает, что размер, форма, характеристики шрифта и цвет элемента преобразуются таким образом, чтобы их внешний вид соответствовал выбранной вами теме оформления jQuery UI. Как показано в листинге 18.1, применение виджетов jQuery UI не доставляет особых хлопот.

Листинг 18.1. Простой HTML-документ

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $.ajax("mydata.json",{
                    success: function(data) {
                        var template = $('#flowerTmpl');
                        template.tmpl(data.slice(0, 3))
                            .appendTo("#row1");
                        template.tmpl(data.slice(3))
                            .appendTo("#row2");
                    }
                });

            $('button').button();
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" data-price="${price}"
                data-stock="${stocklevel}" value="0" required />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                </div>
                <div id="row2"class="drow">
                </div>
            </div>
        </div>
        <div id="buttonDiv"><button type="submit">Заказать
        </button></div>
    </form>
</body>
</html>
```

Применение виджета Button сводится к использованию jQuery для выбора элементов, которые вы хотите преобразовать, и вызову метода `button()`. Все осталь-

ные заботы jQuery UI берет на себя. Результат представлен на рис. 18.1. Обратите внимание на то, что метод `button()` применяется к объекту выбранного набора элементов jQuery. Между библиотеками jQuery и jQuery UI существует очень тесная интеграция, а это означает, что jQuery UI в целом используется так же, как и базовые средства jQuery, рассмотренные в предыдущих частях.

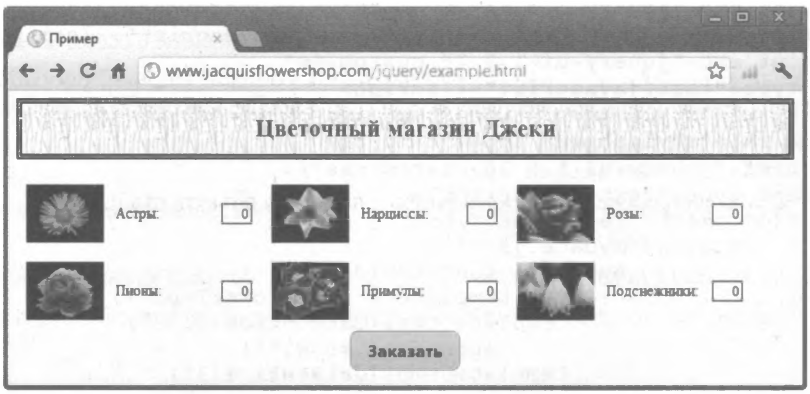


Рис. 18.1. Применение виджета Button

Подобно всем остальным виджетам jQuery UI, виджет Button, который вы видите на рисунке как кнопку, представляет собой набор стилей CSS, примененных к существующему HTML-элементу. В результате применения метода `button()` элемент

```
<button type="submit">Заказать</button>
```

преобразуется в следующий элемент:

```
<button type="submit"
    class="ui-button ui-widget ui-state-default ui-corner-all
           ui-button-text-only" role="button"
           aria-disabled="false">
    <span class="ui-button-text">Заказать</span>
</button>
```

Элегантность этого подхода состоит в том, что он позволяет работать с HTML-элементами привычным способом, не заботясь о том, применены к ним виджеты jQuery UI или нет.

Настройка виджета Button

Виджет Button jQuery UI имеет ряд настраиваемых свойств, с помощью которых можно управлять способом создания результирующей кнопки. Перечень этих свойств приведен в табл. 18.2.

Таблица 18.2. Свойства виджета `Button`

Свойство	Описание
`disabled`	Позволяет определить, отключена ли кнопка, или изменить ее состояние. Отключенной кнопке соответствует значение `true`. Состояние базового HTML-элемента игнорируется в jQuery UI
`text`	Позволяет определить, отображается ли текст кнопки, а также установить или отменить отображение текста. Если значение `icons` равно `false`, то эта опция игнорируется

Окончание табл. 18.2

Свойство	Описание
icons	Позволяет определить, отображаются ли значки в тексте кнопки, а также задать отображаемые значки или отменить их отображение
label	Позволяет получить или изменить текст кнопки

Описанные опции могут применяться двумя способами. Первый способ предполагает использование объекта отображения при вызове метода `button()`, как показано в листинге 18.2.

Листинг 18.2. Конфигурирование виджета Button с помощью объекта отображения

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
                success: function(data, status) {
                    var template = $('#flowerTmpl');
                    template.tmpl(data.slice(0, 3))
                        .appendTo("#row1");
                    template.tmpl(data.slice(3))
                        .appendTo("#row2");
                }
            });

        $('button').button({
            label: "Заказать",
            disabled: true
        });

        $('button').button("option", "disabled", false);
    });
</script>
...
```

Здесь свойство `label` используется для указания текста, который должен отображаться на кнопке, а свойство `disable` — для отключения кнопки. Это соответствует стилю, с которым вы уже встречались при настройке Ajax-запросов, и его следует придерживаться при установке начальных конфигураций виджетов.

В листинге 18.2 продемонстрирован и второй способ, который используется для получения или задания новых значений свойств уже после создания экземпляра виджета.

```
$('button').button("option", "disabled", false);
```

В данном случае также вызывается метод `button()`, но теперь с тремя аргументами. Первый аргумент — это имя метода `option`, второй — свойство, значение которого вы хотите изменить, а третий — новое значение, присваиваемое свойству. Здесь для свойства `disabled` устанавливается значение `false`, которое заменяет значение, установленное перед этим с помощью объекта отображения при создании экземпляра виджета.

Обе методики могут объединяться в одном вызове. В этом случае методу `button()` в качестве первого аргумента передается метод `option`, а в качестве второго —

объект отображения. Это позволяет указать сразу несколько опций при вызове метода `button()`, как показано в листинге 18.3.

Листинг 18.3. Использование аргумента `option` вместе с объектом отображения

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
                success: function(data, status) {
                    var template = $('#flowerTmpl');
                    template.tmpl(data.slice(0, 3))
                        .appendTo("#row1");
                    template.tmpl(data.slice(3))
                        .appendTo("#row2");
                }
            });

        $('button').button()

        $('button').button("option",{
            label: "Заказать",
            disabled: false
        });

        console.log("Кнопка отключена? " + $('button')
           .button("option", "disabled"));
    });
</script>
...
```

Для чтения значения параметра здесь используется все тот же немного "корявый" синтаксис. В данном случае метод `button()` вызывается с двумя аргументами. Первый из них — метод `option`, а второй — свойство, значение которого требуется получить, как показано ниже.

```
console.log("Кнопка отключена? " + $('button')
                              .button("option", "disabled"));
```

Эта инструкция считывает значение свойства `disabled` и выводит его на консоль.

```
Кнопка отключена? false
```

Использование значков jQuery UI на кнопках

Темы jQuery UI включают ряд изображений в виде значков, которые можно использовать для любых целей, в том числе для отображения на кнопках. Пример использования значков на кнопках jQuery UI приведен в листинге 18.4.

Листинг 18.4. Отображение значка на кнопке

```
...
$('button').button({
    icons: {
        primary: "ui-icon-star",
        secondary: "ui-icon-circle-arrow-e"
```

```
    }
});
...
```

Опция `icons` позволяет указать, какие значки следует отображать. В виджете Button для значков предусмотрены две позиции. Свойству `primary` соответствует значок, располагающийся слева от текста, а свойству `secondary` — справа от текста. Как показано в листинге, для указания требуемых значков используется объект со свойствами `primary` и `secondary`. Любое из этих свойств может быть опущено, что приведет к отображению только одного значка. Размеры самих значков очень малы, как показано на рис. 18.2.

Рис. 18.2. Отображение значков на кнопке

Значки идентифицируются с помощью классов, определения которых содержатся в CSS-файле jQuery UI. Общее число доступных значков составляет 173 — слишком много, чтобы здесь можно было привести их полный список. Проще всего выбрать нужный значок, посетив сайт `http://jqueryui.com`. Выберите страницу Themes и перейдите к ее нижней части. Там вы увидите весь набор значков, представленный в виде таблицы (рис. 18.3). При наведении указателя мыши на какой-либо значок отображается имя класса, соответствующего данному значку.

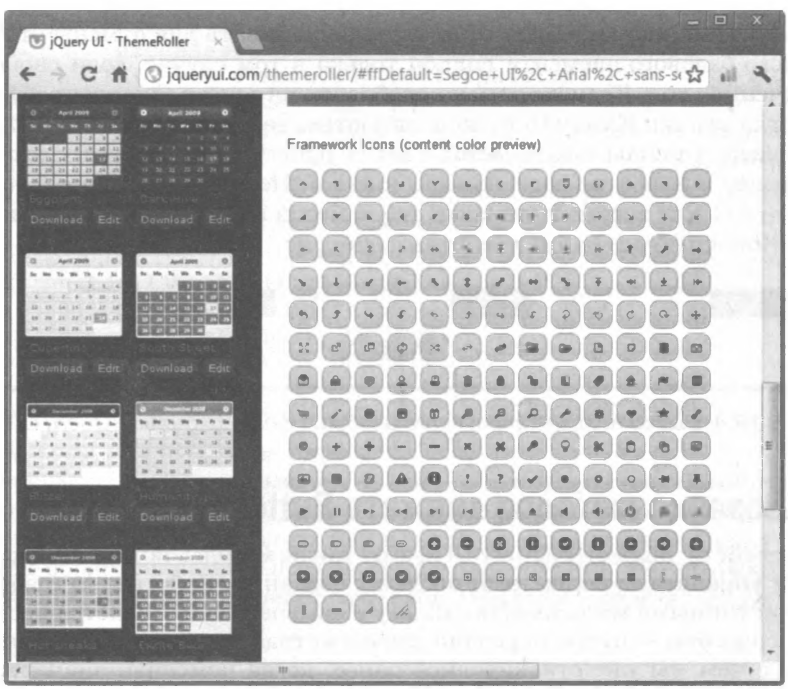


Рис. 18.3. Значки jQuery UI

Совет. Имя значка, появляющееся во всплывающей подсказке, начинается с точки (.), которую при указании значка в опции `icons` следует опускать. Например, если при наведении указателя мыши на первый из значков отображается имя класса `.ui-icon-caret-1-n`, то для того, чтобы использовать на кнопке это изображение, в свойствах `primary` и `secondary` следует указывать значение `ui-icon-caret-1-n`.

Применение пользовательских изображений

Из-за небольшого размера значков jQuery UI я редко использую их в своих приложениях. Для вывода других изображений на кнопках jQuery UI существует несколько способов. Один из них заключается во вставке элемента `img` в тот HTML-элемент `button`, к которому применяется интерфейс jQuery UI. Кнопка jQuery UI должным образом учитывает содержимое базового элемента `button`, и коль скоро вы используете изображение с прозрачным фоном, то вам не нужно заботиться о том, чтобы оно хорошо вписывалось в выбранную вами тему оформления. Простой пример этого приведен в листинге 18.5.

Листинг 18.5. Применение пользовательского изображения на кнопке jQuery UI

```
...
$('button')
    .text("")
    .append("<img src=rightarrows.png width=100 height=30 />")
    .button();
...
```

Свойство `text` кнопки jQuery UI можно использовать для отмены отображения содержимого базового элемента `button` только в том случае, если свойство `icon` имеет значение `true`. Если же отмена отображения текста кнопки требуется в том случае, когда значки jQuery UI не используются, то для получения требуемого результата следует использовать метод `text()` jQuery и установить с его помощью пустую строку в качестве содержимого кнопки. После этого достаточно вызвать метод `append()` для вставки элемента `img` и метод `button()` для создания кнопки jQuery UI. Конечный результат представлен на рис. 18.4.

Рис. 18.4. Вывод пользовательского изображения на кнопке jQuery UI

Использование методов виджета Button

Кроме свойств, виджеты jQuery UI имеют также методы, которые можно использовать для управления виджетами после их создания. Собственно говоря, они не являются истинными методами, поскольку их вызов осуществляется несколько необычным способом — путем передачи имени метода в качестве аргумента методу `button()`, с чем мы уже сталкивались ранее, когда изменяли значения свойств

кнопки с помощью метода option[1]. Тем не менее мы будем называть их методами, поскольку именно такая терминология принята в jQuery UI. Перечень доступных методов вместе с кратким описанием их назначения приведен в табл. 18.3.

Таблица 18.3. Методы виджета Button

Метод	*Описание*
`button("destroy")`	Возвращает базовый элемент в первоначальное состояние, полностью удаляя из него функциональность виджета
`button("disable")`	Отключает кнопку
`button("enable")`	Включает кнопку
`button("option")`	Устанавливает одно или несколько значений свойств
`button("refresh")`	Обновляет состояние кнопки (рассматривается далее).

Удаление виджета

Метод `destroy` удаляет виджет jQuery UI из HTML-элемента, возвращая его в исходное состояние. Соответствующий пример приведен в листинге 18.6.

Листинг 18.6. Использование метода `destroy`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
                success: function(data, status) {
                    var template = $('#flowerTmpl');
                    template.tmpl(data.slice(0, 3))
                        .appendTo("#row1");
                    template.tmpl(data.slice(3))
                        .appendTo("#row2");
                }
            });

        $('button').button().click(function(e) {
            $('button').button("destroy");
            e.preventDefault();
        })
    });
</script>
...
```

В этом примере с помощью метода `click()` регистрируется обработчик щелчка на кнопке. Обратите внимание, что это делается так же, как в главе 9, и не требует принятия каких-либо специальных мер в связи с использованием jQuery UI. Внутри функции-обработчика вызывается метод `destroy`, так что щелчок на кнопке приводит к тому, что она отключает саму себя. Результат представлен на рис. 18.5.

[1] Чтобы методы виджетов можно было легко отличить от других методов jQuery UI, ниже при их описании скобки после имени метода не ставятся. — *Примеч. ред.*

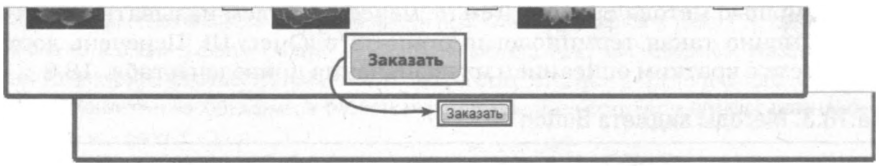


Рис. 18.5. Удаление виджета кнопки jQuery UI

Включение и отключение кнопки

Методы `enable` и `disable` позволяют изменить состояние кнопки jQuery UI, как показано в листинге 18.7.

Листинг 18.7. Включение и отключение кнопки

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
                success: function(data, status) {
                    var template = $('#flowerTmpl');
                    template.tmpl(data.slice(0, 3))
                        .appendTo("#row1");
                    template.tmpl(data.slice(3))
                        .appendTo("#row2");
                }
            });

        $('<span>Включена:<span><input type=checkbox checked />')
            .prependTo('#buttonDiv');
        $(':checkbox').change(function(e) {
            $('button').button(
                $(':checked').length == 1 ? "enable" : "disable"
            )
        });

        $('button').button();
    });
</script>
...
```

В этом сценарии в документ вставлен флажок, и с помощью метода `change()` зарегистрирована функция, которая будет вызываться каждый раз при снятии или установке флажка. Для изменения состояния кнопки в соответствии с состоянием флажка используются методы `enable` и `disable`. Результат представлен на рис. 18.6.

Обновление состояния кнопки jQuery UI

Метод `refresh` обновляет состояние кнопки jQuery UI для учета любых возможных изменений базового HTML-элемента. Этой возможностью удобно пользоваться для отражения изменений, вносимых программным путем, как показано в листинге 18.8.

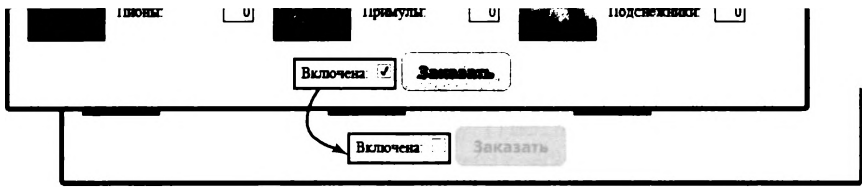


Рис. 18.6. Включение и отключение кнопки jQuery UI

Листинг 18.8. Обновление состояния кнопки jQuery UI

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
                success: function(data, status) {
                    var template = $('#flowerTmpl');
                    template.tmpl(data.slice(0, 3))
                        .appendTo("#row1");
                    template.tmpl(data.slice(3))
                        .appendTo("#row2");
                }
            });

        $('<span>Включена:<span><input type=checkbox checked />')
            .prependTo('#buttonDiv');
        $(':checkbox').change(function(e) {
            var buttons = $('button');
            if ($(':checked').length == 1) {
                buttons.removeAttr("disabled");
            } else {
                buttons.attr("disabled", "disabled");
            }
            buttons.button("refresh");
        });

        $('button').button();
    });
</script>
...
```

В этом примере флажок используется для управления добавлением и удалением атрибута `disabled` из HTML-элемента `button`. Поскольку jQuery UI не обеспечивает автоматического обнаружения этих изменений, то для синхронизации состояний используется метод `refresh`.

Использование событий виджета Button

Кроме свойств и методов, для виджетов jQuery UI определены события, которые можно использовать наряду с событиями, связываемыми с базовыми элементами. Для виджета `button` определено единственное событие — `create`, которое происходит в момент создания кнопки jQuery UI. Как и в случае других методов, работа с событиями ведется с использованием предопределенных аргументов, передаваемых методу jQuery UI, в данном случае — методу `button()`. Пример использования события `create` представлен в листинге 18.9.

Листинг 18.9. Использование события `create` кнопки jQuery UI

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('button').button({
            create: function(e) {
                $(e.target).click(function(ev) {
                    ev.preventDefault();
                    alert("Была нажата кнопка");
                })
            }
        });
    });
</script>
...
```

Здесь событие `create` используется для определения функции, которая будет вызываться в ответ на выполнение щелчка на кнопке. Я не отношу событие `create` к разряду особенно полезных, поскольку считаю, что все то, что может быть выполнено в ответ на возникновение этого события, можно реализовать в рамках более широкого подхода, обеспечиваемого jQuery.

Создание различных типов кнопок

Метод `button()` различает виды элементов, к которым он применяется. Базовое поведение, соответствующее поведению обычной кнопки, создается при вызове метода `button()` для элементов `button`, `a` и `input`, атрибут `type` которых имеет одно из значений `submit`, `reset` или `button`. Пример преобразования всех этих элементов в кнопки jQuery UI приведен в листинге 18.10.

Листинг 18.10. Создание стандартных кнопок

```
...
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            $('.jqButton').click(function(e) {
               e.preventDefault();
               $(this).button();
            });
        });
    </script>
</head>
<body>
    <form>
```

```
        <input class="jqButton" type="submit" id="inputSubmit"
            value="Submit">
        <input class="jqButton" type="reset" id="inputReset"
            value="Reset">
        <input class="jqButton" type="button" id="inputButton"
            value="Input Button">
        <button class="jqButton">Элемент Button</button>
        <a class="jqButton" href="http://apress.com">Элемент A</a>
    </form>
</body>
</html>
...
```

В этом простом документе определены все вышеупомянутые элементы. Функция, переданная методу `click()`, обеспечивает преобразование каждого из элементов в соответствующую кнопку jQuery UI при выполнении щелчка на нем. Результаты такого преобразования представлены на рис. 18.7.

Рис. 18.7. Создание стандартных кнопок jQuery UI

Создание кнопки-переключателя

Вызвав метод `button()` для элемента `input`, типом которого является `checkbox`, вы получите кнопку-переключатель. Эта кнопка может находиться в двух состояниях — "включено" и "выключено" — и поочередно переходит из одного в другое при выполнении на ней щелчков, следуя за сменой состояний "отмечено" и "не отмечено" базового элемента-флажка. Соответствующий пример приведен в листинге 18.11.

Листинг 18.11. Применение jQuery UI к флажку

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('.jqButton').button();
        });
```

```
    </script>
</head>
<body>
    <form>
        <input class="jqButton" type="checkbox" id="toggle"
        <label for="toggle">Переключи меня<label/>
    </form>
</body>
</html>
```

Для создания кнопки jQuery UI на основе флажка требуется элемент `input` с соответствующим элементом `label`, как показано в листинге. Создаваемая кнопка-переключатель выглядит так же, как и обычная кнопка jQuery UI, но поочередно переходит в одно из двух возможных состояний при выполнении щелчков на ней. Результат представлен на рис. 18.8.

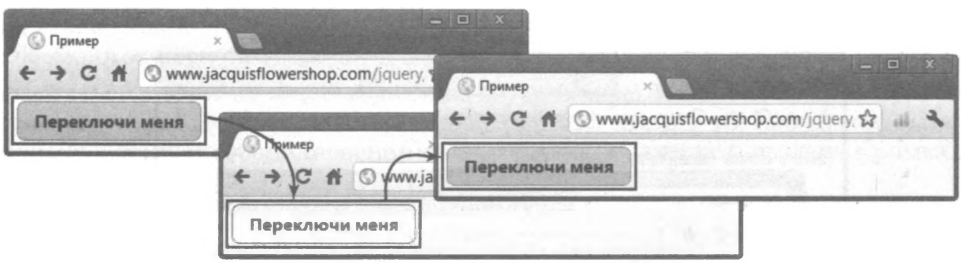


Рис. 18.8. Создание переключателя на основе флажка

Не забывайте о том, что jQuery UI не изменяет базовые элементы, и поэтому исходный элемент-флажок сохраняет свою функциональность, а его состояние по-прежнему контролируется атрибутом `checked`, как если бы средства jQuery UI не были применены.

Создание группы переключателей

Метод `buttonset()` позволяет объединить ряд взаимозависимых переключателей (радиокнопок, т.е. элементов `input` с типом `radio`) в группу jQuery UI, как показано в листинге 18.12.

Листинг 18.12. Создание группы переключателей jQuery UI

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#radiodiv').buttonset();
        });
    </script>
</head>
```

```
<body>
    <form>
        <div id="radioDiv">
            <input type="radio" name="flower" id="rose" checked />
                <label for="rose">Розы</label>
            <input type="radio" name="flower" id="lily"/>
                <label for="lily">Лилии</label>
            <input type="radio" name="flower" id="iris"/>
                <label for="iris">Ирис</label>
        </div>
    </form>
</body>
</html>
```

Обратите внимание на то, что метод `buttonset()` вызывается для предварительно выбранного набора переключателей (радиокнопок), содержащихся в контейнерном элементе `div`. В данной ситуации вы не должны вызывать метод `button()` для каждого из элементов `input` по отдельности. Результат применения метода `buttonset()` представлен на рис. 18.9.

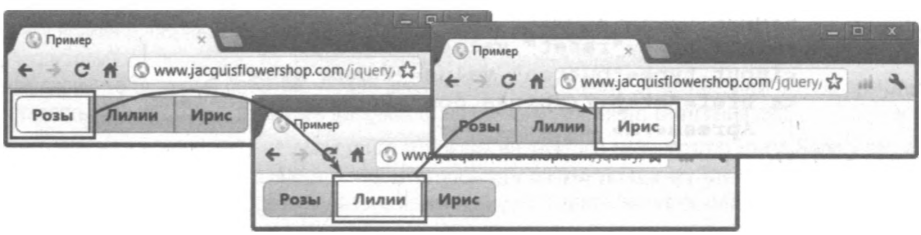


Рис. 18.9. Создание группы переключателей

Как и в случае обычных переключателей, в любой момент может быть выбрана только одна кнопка, что дает возможность предоставить пользователю фиксированный набор вариантов выбора, стилевое оформление которого согласуется с оформлением других кнопок jQuery UI. Заметьте, что jQuery UI учитывает тот факт, что кнопки взаимосвязаны, применяя к стыкующимся краям кнопок иной стиль оформления, нежели к наружным. Это хорошо видно на рис. 18.10.

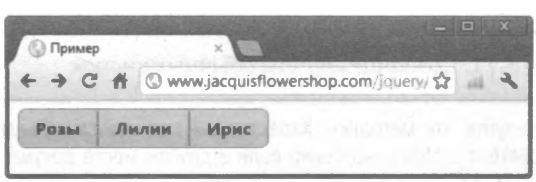


Рис. 18.10. Стилизация набора кнопок jQuery UI

Создание группы обычных кнопок jQuery UI

Метод `buttonset()` можно использовать по отношению к любым другим элементам, к которым применим метод `button()`. Конечным результатом этого является применение *стиля* группы переключателей, но не *поведения*, поэтому каждая кнопка будет работать независимо от других. Пример такого варианта использования метода `buttonset()` представлен в листинге 18.13.

Листинг 18.13. Создание группы обычных кнопок jQuery UI

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#radiodiv').buttonset();
        });
    </script>
</head>
<body>
    <form>
        <div id="radioDiv">
            <input type="submit" value="Submit"/>
            <input type="reset" value="Reset"/>
            <input type="button" value="Нажми меня"/>
            <a href="http://apress.com">Посетите веб-сайт
               Apress</a>
        </div>
    </form>
</body>
</html>
```

В кнопку преобразуется любой подходящий элемент, находящийся внутри контейнера `div`, а стыкующиеся края кнопок стилизуются точно так же, как и в случае переключателей, как показано на рис. 18.11.

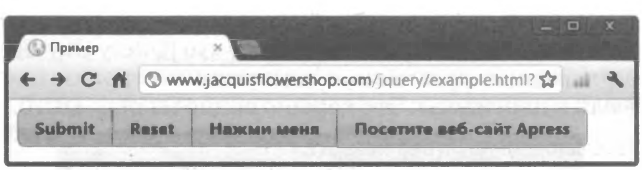


Рис. 18.11. Создание группы обычных кнопок

Совет. Осторожно используйте эту методику. Характерный вид кнопок, объединенных в одну группу, может сбивать пользователя с толку, особенно если в другом месте документа или веб-приложения есть другая группа кнопок, которая действительно выполняет функции переключателя.

Использование виджета Progress Bar

Теперь, когда вы получили общее представление о виджетах jQuery UI на примере виджета Button, мы можем приступить к рассмотрению остальных виджетов, начав с виджета Progress Bar — индикатора процесса.

С помощью индикатора процесса можно информировать пользователя о достигнутом прогрессе в ходе выполнения какой-либо задачи. Этот индикатор используется

лишь в случае *детерминированных* задач, т.е. задач, масштаб которых известен, и можно точно определить процентную долю выполненного объема работы. Другую категорию составляют *недетерминированные* задачи. Ход выполнения таких задач с трудом поддается количественной оценке, так что в этом случае необходимо лишь сообщить пользователю, что до завершения задачи придется немного подождать (простой пример такого индикатора в случае недетерминированного процесса, когда для этой цели использовалось анимированное изображение, приводился в главе 16).

Отображение полезной информации о ходе выполнения задач

Никаких жестких правил относительно того, каким образом виджеты должны использоваться в веб-приложениях, не существует. В то же время в пользовательской среде, мнение которой формируется под влиянием стандартов, устанавливаемых такими операционными системами, как Windows или Mac OS, уже сложились определенные представления о том, каким должно быть поведение тех или иных элементов управления, и в частности индикатора процесса. Чтобы использование индикатора процесса приносило действительную пользу, придерживайтесь следующих правил.

Во-первых, изменяйте значение индикатора лишь в сторону увеличения. Не пытайтесь уменьшать его, если оказывается, что для завершения задачи требуется выполнить большее число действий, чем первоначально предполагалось. Индикатор процесса должен отражать степень выполнения задачи, а не оценку времени, оставшегося до ее завершения. Если существует несколько возможных путей развития вычислительного процесса, то значение индикатора должно базироваться на наиболее пессимистическом варианте развития событий. Лучше впоследствии отобразить существенный скачок значения, чем заставлять пользователя теряться в догадках относительно реального состояния дел.

Во-вторых, ни в коем случае не допускайте хождения индикатора процесса по кругу. Если у вас имеется недостаточно информации для того, чтобы отобразить разумную оценку степени выполнения задачи, используйте индикатор процесса, предназначенный для недетерминированных задач. Если значение индикатора приближается к отметке 100%, то пользователь настраивается на то, что процесс вот-вот закончится. Если же индикатор внезапно возвращается в начало шкалы, то это просто-напросто сбивает пользователя с толку и делает применение индикатора процесса бессмысленным.

Создание виджета Progress Bar

Чтобы создать индикатор процесса, следует выбрать элемент `div` и вызвать метод `progressbar()`, как показано в листинге 18.14.

Листинг 18.14. Создание индикатора процесса

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#progressDiv').progressbar({
                value: 21
            });
        });
    </script>
```

```
</head>
<body>
        <div id="progressDiv"></div>
</body>
</html>
```

В этом примере документ содержит элемент `div` с идентификатором `progressDiv`. Для создания индикатора процесса следует использовать пустой элемент `div`. Наличие любого содержимого влияет на расположение виджета на странице. В сценарии мы выбираем элемент `progresDiv` и вызываем метод `progressbar()`, передавая ему объект отображения с настройками начальной конфигурации индикатора. Виджет Progress Bar поддерживает два настраиваемых свойства, которые описаны в табл. 18.4.

Таблица 18.4. Свойства виджета Progress Bar

Свойство	*Описание*
`disabled`	Значение `true` соответствует отключенному индикатору процесса. Значение по умолчанию — `false`
`value`	Устанавливает процент выполнения задачи, отображаемый для пользователя. Значение по умолчанию — 0

В данном примере начальное значение индикатора установлено равным `21%`. Результат представлен на рис. 18.12.

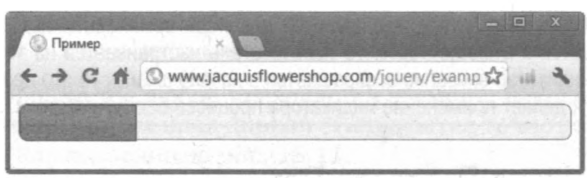


Рис. 18.12. Создание индикатора процесса

Использование методов виджета Progress Bar

Для виджета Progress Bar определен ряд методов, для которых используется та же форма вызова, что и для методов виджета Button. Иными словами, вы вызываете метод `progressbar()` и в качестве первого аргумента указываете требуемый метод. Доступные методы представлены в табл. 18.5.

Таблица 18.5. Методы виджета Progress Bar

Метод	*Описание*
`progressbar("destroy")`	Возвращает элемент `div` в исходное состояние, полностью удаляя из него функциональность виджета
`progressbar("disable")`	Отключает индикатор процесса
`progressbar("enable")`	Включает индикатор процесса
`progressbar("option")`	Устанавливает одно или несколько значений свойств
`progressbar("value", value)`	Получает или устанавливает значение, отображаемое индикатором процесса

Большинство этих методов работает аналогично одноименным методам виджета Button, поэтому я не буду терять время на пояснение их примерами. Исключение составляет метод `value`, который позволяет получать или устанавливать значение, отображаемое индикатором процесса. Соответствующий пример приведен в листинге 18.15.

Листинг 18.15. Использование метода `value` виджета Progress Bar

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#progressDiv').progressbar({
                value: 21
            });

            $('button').click(function(e) {
                var divElem = $('#progressDiv');
                var currentProgress =
                        divElem.progressbar("value");
                    divElem.progressbar("value",
                        this.id == "decr" ?
                            currentProgress - 10 :
                                currentProgress + 10)
            })
        })
    </script>
</head>
<body>
    <div id="progressDiv"></div>

    <button id="decr">Уменьшить</button>

    <button id="incr">Увеличить</button>
</body>
</html>
```

В этом примере добавлена пара элементов `button`, которые используются для уменьшения или увеличения значения, отображаемого индикатором процесса. Каждое нажатие кнопки увеличивает значение на 10%. Результат представлен на рис. 18.13.

Совет. Метод `value` всегда возвращает значение, лежащее в интервале от `0` до `100`, даже если вы установите значение, выходящее за эти пределы в ту или иную сторону. Это означает, что проверку указываемых вами значений можно возложить на индикатор, а не заниматься этим самому.

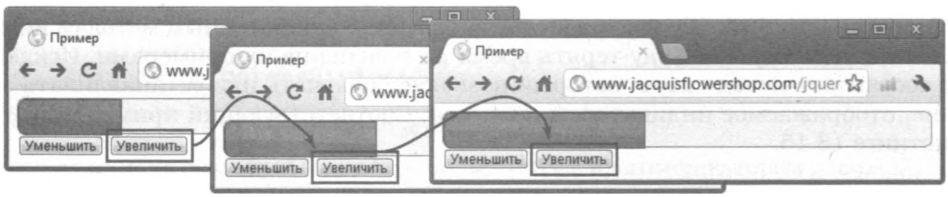


Рис. 18.13. Использование метода `value` *для изменения значения, отображаемого индикатором*

Анимация индикатора процесса

Несмотря на то что внешний вид индикатора процесса согласуется с используемой темой оформления jQuery UI, он выглядит совсем просто. При создании этого индикатора jQuery UI добавляет в документ элемент `div`, а также определяет ряд новых классов как в новом элементе `div`, так и в том, для которого вызывался метод `progressbar()`. Сгенерированная HTML-разметка выглядит примерно следующим образом.

```
<div id="progressDiv" class=
    "ui-progressbar ui-widget ui-widget-content ui-corner-all"
    role="progressbar" aria-valuemin="0" aria-valuemax="100"
    aria-valuenow="10">
    <div class="ui-progressbar-value ui-widget-header
        ui-corner-left" style="width: 10%; ">
    </div>
</div>
```

Класс `ui-progressbar-value` воздействует на элемент, который jQuery UI добавляет для отображения значения индикатора, а класс `ui-progressbar` — на внешний элемент `div`, от которого мы отталкивались. Эти классы можно привлечь для создания индикатора процесса, в котором используется анимированное GIF-изображение, как показано в листинге 18.16.

Листинг 18.16. Использование анимированного GIF-изображения в индикаторе процесса

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .ui-progressbar-value {
            background-image: url(progress-animation.gif);
        }
        .ui-progressbar {
            height: 22px
        }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
```

```
            $('#progressDiv').progressbar({
                value: 75
            });
        })
    </script>
</head>
<body>
    <div id="progressDiv"></div>

</body>
</html>
```

Для указания изображения, которое будет использоваться внутренним элементом `div`, можно воспользоваться CSS-свойством `background-image`. В данном случае указано изображение `progress-animation.gif`, которое представляет собой простое анимированное GIF-изображение, взятое на сайте jQuery UI. С помощью снимка экрана трудно передать эффект анимации, однако один из ее кадров представлен на рис. 18.14.

Рис. 18.14. Использование анимированного изображения в индикаторе процесса

По поводу использования изображений, подобных этому, можно сделать два замечания. Во-первых, ответственность за то, чтобы выбранное изображение согласовывалось с остальной частью темы, лежит на вас. Во-вторых, необходимо следить за высотой изображения. По умолчанию высота индикатора процесса jQuery UI составляет `2em`, что может породить проблемы в случае меньших изображений. Чтобы эти трудности не возникали, установите для свойства `height` элементов класса `ui-progressbar` значение, совпадающее с высотой изображения, которое используется. В данном примере высота изображения составляет 22 пикселя. Если не предпринять никаких мер по регулированию высоты, то границы шкалы индикатора либо окажутся спрятанными за изображением, либо будут выступать за его пределы, как показано на рис. 18.15.

Рис. 18.15. Изображение, высота которого составляет менее `2em`

Использование событий виджета Progress Bar

Для индикатора процесса jQuery UI определены три события, которые описаны в табл. 18.6.

Таблица 18.6. События виджета Progress Bar

Событие	*Описание*
`create`	Происходит при создании индикатора процесса
`change`	Происходит при изменении значения, отображаемого индикатором процесса
`complete`	Происходит, когда значение, отображаемое индикатором процесса, достигает отметки `100`

Пример использования событий приведен в листинге 18.17.

Листинг 18.17. Использование событий виджета Progress Bar

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .ui-progressbar-value {
            background-image: url(progress-animation.gif);
        }
        .ui-progressbar {
            height: 22px
        }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('button').button();

            $('#progressDiv').progressbar({
                value: 75,
                create: function(e) {
                    $('#progVal').text($('#progressDiv')
                        .progressbar("value"));
                },
                complete: function(e) {
                    $('#incr').button("disable")
                },
                change: function(e) {
                    if ($(this).progressbar("value") < 100) {
                        $('#incr').button("enable"
                    }
                    $('#progVal').text($('#progressDiv')
                        .progressbar("value"));
                }
            });

            $('button').click(function(e) {
                var divElem = $('#progressDiv');
                var currentProgress =
```

```
                    divElem.progressbar("value");
                divElem.progressbar("value",
                    this.id == "decr" ? .
                        currentProgress - 10 :
                            currentProgress + 10)
            })
        });
    </script>
</head>
<body>
    <div id="progressDiv"></div>
    <button id="decr">Уменьшить</button>
    <button id="incr">Увеличить</button>
    Ход процесса: <span id="progVal"></span>%
</body>
</html>
```

В этом примере добавлен элемент span, который используется для отображения процентного значения индикатора процесса. Результат представлен на рис. 18.16.

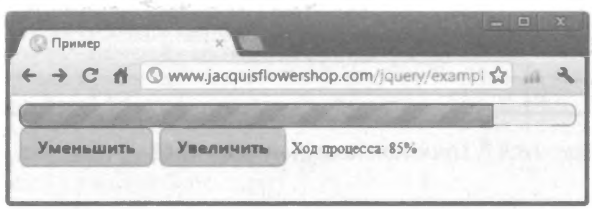


Рис. 18.16. Реагирование на события индикатора процесса

Совет. Работая с событиями, следует помнить о двух вещах. Во-первых, событие `complete` срабатывает всякий раз, когда индикатор достигает отметки `100` или переходит ее. Это означает, что в случае многократной повторной установки значения индикатора равным, например, `100`, событие может запускаться множество раз. Во-вторых, при значения индикатора, равном `100` и выше, запускаются два события — `change` и `complete`, поэтому нужно быть готовым обработать оба события, когда завершаете обновление индикатора процесса.

Использование виджета Slider

Виджет Slider позволяет создавать ползунки из элементов HTML-документа. Для создания ползунков используется метод `slider()`, как показано в листинге 18.18. Ползунки применяются в тех случаях, когда пользователю необходимо предоставить возможность выбирать значения, лежащие в некотором заданном диапазоне.

Листинг 18.18. Создание ползунка

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
```

```
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#slider').slider();

        });
    </script>
</head>
<body>
    <div id="slider"></div>
</body>
</html>
```

Визуальный стиль ползунка автоматически подстраивается под тему оформления, общую для всех виджетов. Пользователь может перемещать ползунок вдоль шкалы с помощью мыши или клавиатуры. Внешний вид базового ползунка показан на рис. 18.17.

Рис. 18.17. Простой ползунок jQuery UI

Настройка виджета Slider

Как и в случае других виджетов jQuery UI, для виджета Slider определен ряд свойств, которые можно использовать для настройки его внешнего вида и поведения. Их краткое описание приведено в табл. 18.7. В следующих разделах будет показано, как использовать эти свойства для настройки виджета.

Таблица 18.7. Свойства виджета Slider

Свойство	*Описание*
`animate`	Значение `true` разрешает анимацию ползунка после выполнения пользователем щелчка на шкале вне рукоятки. Значение по умолчанию — `false`
`disabled`	Значение `true` соответствует отключенному состоянию ползунка. Значение по умолчанию — `false`
`max`	Определяет максимальное значение для ползунка. Значение по умолчанию — `100`
`min`	Определяет минимальное значение для ползунка. Значение по умолчанию — `0`
`orientation`	Определяет ориентацию ползунка (см. пример далее)
`range`	Используется совместно со свойством `values` для создания ползунка с несколькими рукоятками
`step`	Определяет шаг перемещения рукоятки вдоль шкалы между минимальным и максимальным значениями
`value`	Определяет значение, представляемое ползунком
`values`	Используется совместно со свойством `range` для создания ползунка с несколькими рукоятками

Совет. Значения `min` и `max` не входят в число допустимых. Таким образом, если вы установите `min` равным `0`, а `max` — равным `100`, то пользователь сможет выбирать значения от `1` до `99`.

Изменение ориентации ползунка

По умолчанию ползунки располагаются в горизонтальном направлении, но, используя свойство `orientation`, можно создавать и вертикальные ползунки.

Простой пример приведен в листинге 18.19.

Листинг 18.19. Использование свойства `orientation`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #hslider, #vslider { margin: 10px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#hslider').slider({
                value: 35
            });

            $('#vslider').slider({
                orientation: "vertical",
                value: 35
            })

        });
    </script>
</head>
<body>
    <div id="hslider"></div>
    <div id="vslider"></div>
</body>
</html>
```

В этом примере создаются два ползунка, для одного из которых свойство `orientation` задано равным `vertical`. В документ добавлен элемент `style`, устанавливающий поля для элементов ползунка, чтобы они располагались на некотором расстоянии друг от друга. Размером и положением ползунков (как и любого другого виджета jQuery UI) можно управлять, применяя стиль к базовому элементу (именно поэтому для создания ползунков лучше всего подходят элементы `div`, которыми легко манипулировать с помощью стилей CSS). Результирующие ползунки изображены на рис. 18.18. Обратите внимание, что начальные позиции рукояток установлены с помощью свойства `value`.

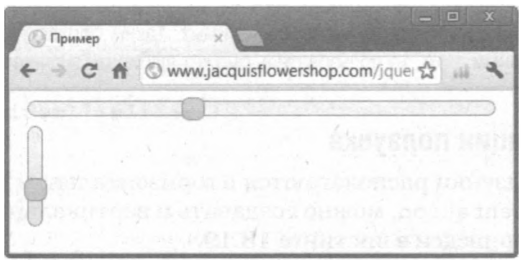


Рис. 18.18. Создание вертикального и горизонтального ползунков

В этом примере ползунки создаются и настраиваются раздельно. Мы можем более эффективно использовать базовую функциональность jQuery, переписав код так, как показано в листинге 18.20.

Листинг 18.20. Более эффективное применение jQuery

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #hslider, #vslider { margin: 10px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#hslider, #vslider').slider({
                value: 35,
                orientation: "vertical"
            }).filter('#hslider')
              .slider('option', 'orientation', 'horizontal');
        });
    </script>
</head>
<body>
    <div id="hslider"></div>
    <div id="vslider"></div>
</body>
</html>
```

Этот вопрос является второстепенным, но мне хотелось, чтобы вы не забывали о том, что библиотека jQuery UI является надстройкой библиотеки jQuery и тесно интегрирована с ней, так что можете свободно пользоваться всеми способами выбора элементов и манипуляций ими, с которыми вы познакомились в предыдущих главах.

Совет. Обратите внимание на то, что сначала создаются два вертикальных ползунка, а затем ориентация одного из них изменяется на горизонтальную. На момент написания книги наблюдался один незначительный программный дефект, заключающийся в том, что рукоятка вертикального ползунка, если первоначально он создавался как горизонтальный, а затем преобразовывался в вертикальный, оказывалась смещенной от центра шкалы[2].

Анимация ползунка

Свойство `animate` позволяет задать плавное перемещение рукоятки ползунка из ее текущего положения в точку шкалы, в которой пользователь выполнил щелчок (в отличие от непосредственного перемещения рукоятки с помощью мыши). Можно разрешить анимацию, предусмотренную по умолчанию, установив для свойства `animate` значение `true`. Кроме того, можно задать скорость анимации с помощью предустановленных строковых значений `fast` и `slow` или путем указания времени (в миллисекундах), в течение которого должна длиться анимация. Соответствующий пример приведен в листинге 18.21.

Листинг 18.21. Использование свойства `animate`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #slider {margin: 10px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#slider').slider({
                animate: "fast"
            });
        });
    </script>
</head>
<body>
    <div id="slider"></div>
</body>
</html>
```

В этом примере для свойства `animate` установлено значение `fast`. Передать анимацию с помощью снимка экрана нелегко, однако рис. 18.19 позволяет получить некоторое представление о том, что при этом происходит.

На этом экранном снимке изображен ползунок в момент, непосредственно предшествующий выполнению на нем щелчка. Если бы анимация не была разрешена, то перемещение рукоятки в точку щелчка осуществлялось бы скачком. Но поскольку мы разрешили анимацию, то перемещение рукоятки из начального положения в конечное будет плавным.

[2] Такое же поведение ползунка наблюдается и в случае использования следующей комбинации библиотек: *jquery-1.7.2 + jquery-ui-1.8.21. — Примеч. ред.*

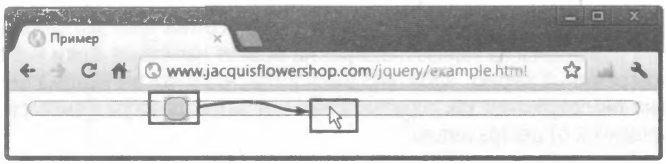


Рис. 18.19. Анимация перемещения рукоятки ползунка

Тем не менее (и это универсальный рецепт для любого эффекта и анимации) главное — не переусердствовать, и именно поэтому я выбрал параметр `fast` (быстро). Это как раз тот случай, когда с примером имеет смысл немного поэкспериментировать для выбора оптимального варианта. Могу порекомендовать загрузить код примера с сайта книги (см. главу 1), чтобы не вводить его вручную.

Создание диапазонного ползунка

Диапазонный ползунок имеет две рукоятки, с помощью которых можно задавать диапазон значений. Скажем, вы могли решить, что было бы неплохо предоставить пользователю возможность самостоятельно указать диапазон цен, по которым он согласился бы купить реализуемую продукцию, что позволило бы исключить из рассмотрения товары, не соответствующие его пожеланиям. Пример создания диапазонного ползунка приведен в листинге 18.22.

Листинг 18.22. Создание диапазонного ползунка

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #slider { margin: 10px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#slider').slider({
                values: [35, 65],
                range: true,
                create: displaySliderValues,
                slide: displaySliderValues
            })

            function displaySliderValues() {
                $('#lower').text($('#slider')
                            .slider("values", 0));
                $('#upper').text($('#slider')
                            .slider("values", 1));
            }
        });
    </script>
```

```
</head>
<body>
    <div id="slider"></div>
    <div>Нижняя граница: <span id="lower">
        </span> Верхняя граница: <span id="upper"></span></div>
</body>
</html>
```

Чтобы создать диапазонный ползунок, необходимо установить значение свойства `range` равным `true` и задать в качестве значения свойства `values` массив, содержащий начальные значения нижней и верхней границ диапазона. (Когда используется обычный ползунок, применяется свойство `value`, а когда используется диапазонный ползунок, применяется свойство `values`.) В данном примере в качестве нижней и верхней границ установлены соответственно значения `35` и `65`. Результат представлен на рис. 18.20.

Рис. 18.20. Создание диапазонного ползунка

Здесь в сценарий добавлена функция-обработчик для событий `create` и `slide`. О событиях, поддерживаемых ползунком, мы еще будем говорить, но сейчас я просто хотел продемонстрировать, как определить позиции рукояток диапазонного ползунка. Для этого используется метод `values()`, которому передается индекс интересующей вас рукоятки, как показано ниже.

```
$('#slider').slider("values", 0)
```

Индексы отсчитываются от нуля, поэтому в приведенном фрагменте считывается значение для рукоятки, представляющей нижнюю границу диапазона. События используются для определения содержимого двух элементов `span`.

Использование методов виджета Slider

Для ползунка определен тот же набор базовых методов, что и для любого другого виджета jQuery UI, плюс еще два метода, позволяющие указать значение или диапазон значений, которые должны отображаться. Эти методы приведены в табл. 18.8.

Таблица 18.8. Методы виджета Slider

Метод	*Описание*
`slider("destroy")`	Возвращает базовый элемент в первоначальное состояние
`slider("disable")`	Отключает ползунок
`slider("enable")`	Включает ползунок
`slider("option")`	Устанавливает одно или несколько значений свойств
`slider("value", значение)`	Получает и устанавливает значение для обычного ползунка
`slider("values", значения)`	Получает и устанавливает значения для диапазонного ползунка

Пример использования методов value и values для управления ползунком из программы приведен в листинге 18.23.

Листинг 18.23. Программное управление ползунками

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #slider, #rangeslider, *.inputDiv { margin: 10px}
        label {width: 80px; display: inline-block; margin: 4px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#slider').slider({
               value: 50,
               create: function() {
                    $('#slideVal')
                        .val($('#slider').slider("value"));
               }
            });

            $('#rangeslider').slider({
                values: [35, 65],
                range: true,
                create: function() {
                    $('#rangeMin').val($('#rangeslider')
                        .slider("values", 0));
                    $('#rangeMax').val($('#rangeslider')
                        .slider("values", 1));
                }
            })

            $('input').change(function(e) {
                switch (this.id) {
                    case "rangeMin":
                    case "rangeMax":
                        var index = (this.id == "rangeMax") ?
                            1 : 0;
                        $('#rangeslider')
                            .slider("values", index, $(this)
                            .val())
                        break;
                    case "slideVal":
                        $('#slider')
                            .slider("value", $(this).val())
                        break;
                }
            })
```

```
        });
    </script>
</head>
<body>
    <div id="rangeslider"></div>
    <div class="inputDiv">
        <label for="rangeMin">Мин. значение: </label>
            <input id="rangeMin" />
        <label for="rangeMax">Макс. значение: </label>
            <input id="rangeMax" />
    </div>
    <div id="slider"></div>
    <div class="inputDiv">
        <label for="slideVal">Значение индикатора: </label>
            <input id="slideVal" />
    </div>
</body>
</html>
```

В этом документе два ползунка. Кроме того, в нем имеются три поля ввода, позволяющие устанавливать нужные значения, не перемещая рукоятки ползунков. Вид документа в окне браузера представлен на рис. 18.21.

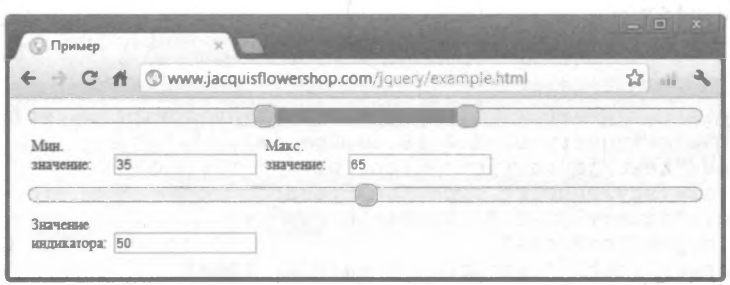


Рис. 18.21. Управление ползунками программным путем

Здесь к элементам `input`, выбранным с помощью jQuery, применяется метод `change()`, в результате чего всякий раз, когда изменяется значение в одном из полей, вызывается указанная функция. Внутри этой функции с помощью оператора `switch` организуется ветвление по значению атрибута `id` измененного элемента, и с помощью метода `value` или `values` устанавливаются позиции его рукояток. Указанная связь между полями ввода и ползунками является односторонней в том смысле, что перемещение рукояток не приводит к изменению значений, введенных в полях. О том, как сделать такую связь двухсторонней, будет рассказано в следующем разделе при рассмотрении событий, поддерживаемых ползунком.

Использование событий виджета Slider

В табл. 18.9 приведены события, поддерживаемые ползунком. Замечательно то, что в число этих событий входят как `change`, так и `stop`, что позволяет различать случаи, когда новые значения создаются в результате перемещения рукоятки ползунка пользователем и когда они устанавливаются программным путем.

Таблица 18.9. События ползунка

Событие	*Описание*
`create`	Происходит в момент создания ползунка
`start`	Происходит, когда пользователь начинает перемещать рукоятку ползунка
`slide`	Происходит при любом движении мыши, приводящем к перемещению ползунка
`change`	Происходит, когда пользователь перестает перемещать рукоятку ползунка или когда значение ползунка изменяется программным путем
`stop`	Происходит, когда пользователь перестает перемещать рукоятку ползунка

Пример использования событий ползунка для создания двухсторонней связи между ползунками и полями ввода в примере из предыдущего раздела приведен в листинге 18.24. Это позволяет связать в единое целое возможности управления ползунками, обеспечиваемые программой и действиями пользователя.

Листинг 18.24. Использование событий ползунка для создания двухсторонней связи между ползунками и полями ввода

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #rangeslider, *.inputDiv { margin: 10px}
        label {width: 80px; display: inline-block; margin: 4px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#rangeslider').slider({
                values: [35, 65],
                range: true,
                create: setInputsFromSlider,
                slide: setInputsFromSlider,
                stop: setInputsFromSlider
            })

            function setInputsFromSlider() {
                $('#rangeMin').val($('#rangeslider')
                    .slider("values", 0));
                $('#rangeMax').val($('#rangeslider')
                    .slider("values", 1));
            }

            $('input').change(function(e) {
                var index = (this.id == "rangeMax") ? 1 : 0;
                $('#rangeslider')
                    .slider("values", index, $(this).val())
```

```
            })
        });
    </script>
</head>
<body>
    <div id="rangeslider"></div>
    <div class="inputDiv">
        <label for="rangeMin">Мин. значение: </label>
            <input id="rangeMin" />
        <label for="rangeMax">Макс. значение: </label>
            <input id="rangeMax" />
    </div>
</body>
</html>
```

Для упрощения примера я удалил один из ползунков. Весь остальной код у нас имелся, поскольку ранее мы уже устанавливали значения в полях ввода в ответ на событие `create`. Чтобы все это могло работать с другими событиями, я выделил необходимые инструкции в отдельную функцию и использовал ее для обработки событий `create`, `slide` и `stop`. Теперь рукоятки ползунка перемещаются при вводе новых значений в полях, а значения в полях ввода обновляются при перемещении рукояток. Вид окончательного документа в окне браузера показан на рис. 18.22, а полное представление о том, как работает данный пример, можно получить только в процессе реального взаимодействия с ползунком.

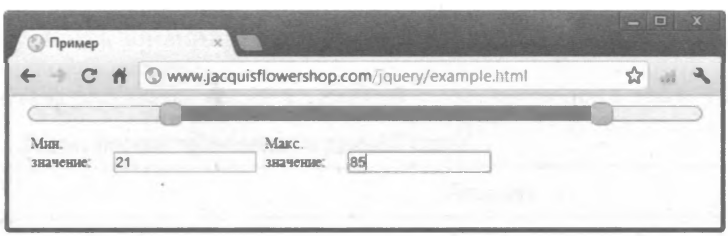


Рис. 18.22. Реагирование на события ползунка

Резюме

В этой главе вы познакомились с первыми тремя виджетами jQuery UI: кнопкой, индикатором процесса и ползунком. Все они имеют одну и ту же базовую структуру. Виджеты создаются и настраиваются с использованием единого метода, также позволяющего предоставлять функцию, которая должна реагировать на события. Есть методы и события, являющиеся общими для всех виджетов, но есть и уникальные дополнительные методы, обеспечивающие специфическую функциональность, свойственную только данному виджету. Теперь, когда вы уже разобрались с основами, можем перейти к изучению более гибких и сложных виджетов, которые рассматриваются в следующих главах.

ГЛАВА 19

Использование виджетов Autocomplete и Accordion

В этой главе описаны виджеты Autocomplete и Accordion библиотеки jQuery UI. В какой-то мере их можно считать более сложными, чем виджеты, рассмотренные в главе 18, но построены они по тому же общему для всех виджетов jQuery UI образцу и обладают своими наборами настраиваемых свойств (опций), методов и событий. Виджеты — это тщательно спроектированные готовые элементы управления с гибкими возможностями, которые при умелом применении позволяют значительно улучшить внешний вид документов и веб-приложений и повысить удобство их использования.

Перечень тем, рассматриваемых в данной главе, приведен в табл. 19.1.

Таблица 19.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Добавление функциональности Autocomplete jQuery UI в элемент `input`	Используйте метод `autocomplete()`	1, 2
Использование удаленного сервера в качестве источника подходящих значений для автозаполнения	Укажите URL-адрес в опции `source`	3–5
Динамическая генерация подходящих значений для автозаполнения	Укажите функцию в опции `source`	6
Программное управление функцией автозаполнения	Используйте методы `search` и `close`	7
Получение уведомления о выбранном элементе списка автозаполнения	Используйте события `focus`, `select` и `change`	8
Замена действий, выполняемых по умолчанию при выборе элемента списка автозаполнения	Замените действие по умолчанию для события `select`	9
Создание виджета Accordion jQuery UI	Используйте метод `accordion()`	10
Установка высоты панелей содержимого виджета Accordion в зависимости от размера содержимого	Используйте опцию `autoHeight`	11, 12
Заполнение виджетом Accordion пространства родительского элемента по всей высоте	Используйте метод `fillSpace()`	13

Окончание табл. 19.1

Задача	Решение	Листинг
Изменение действия, которое пользователь должен выполнить для активизации (открытия) панели содержимого	Используйте опцию `event`	14
Задание панели содержимого, которая должна быть активизирована сразу после создания виджета Accordion	Используйте опции `active` и `collapsible`	15, 16
Изменение значков, используемых в виджете Accordion	Используйте опцию `icons`	17
Переключение панелей содержимого виджета Accordion *программным способом*	Используйте метод `activate`	18
Получение уведомлений о переключении панелей содержимого виджета Accordion	Обработайте события `change` и `changestart`	19

Использование виджета Autocomplete

Виджет Autocomplete облегчает ручной ввод информации, предлагая на выбор пользователю варианты автоматического заполнения текстовых полей по мере ввода символов. Продуманное использование этого виджета обеспечивает значительную экономию времени за счет ускорения ввода данных и снижения вероятности ошибок ввода. В следующем разделе показано, как создавать, настраивать и использовать виджет Autocomplete.

Создание виджета Autocomplete

Чтобы создать элемент управления автозаполнением, следует вызвать метод `autocomplete()` для соответствующего элемента `input` Пример использования этого метода, в котором демонстрируется базовая функциональность виджета Autocomplete, приведен в листинге 19.1.

Листинг 19.1. Создание текстового поля с функцией автозаполнения

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            var flowers = ["Астры", "Нарциссы", "Розы", "Пионы",
                "Примулы", "Подснежники", "Маки", "Первоцветы",
                "Петунии", "Фиалки"];

            $('#acInput').autocomplete({
                source: flowers
            })
        });
```

```
    </script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Название цветов: </label>
            <input id="acInput"/>
        </div>
    </form>
</body>
</html>
```

Метод `autocomplete()` работает аналогично любому другому методу jQuery UI из тех, с которыми вы уже познакомились, за исключением того, что ему необходимо передавать объект отображения, содержащий значение опции `source`. Эта опция позволяет указать источник, из которого должны извлекаться данные для автозаполнения. В примере роль такого источника играет простой массив значений. На рис. 19.1 показано, в каком виде средство автозаполнения предстает перед пользователем.

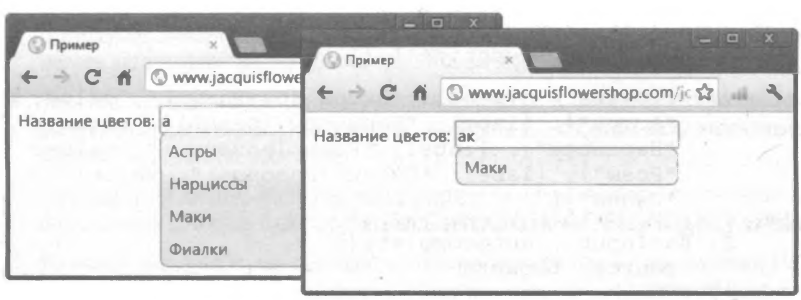


Рис. 19.1. Базовая реализация функции автозаполнения

На этом рисунке представлены два экранных снимка. Первый из них соответствует ситуации, когда пользователь вводит букву *а*. Вы видите, что это приводит к появлению списка цветов, названия которых включают букву *а*. В список вошли не только астры, название которых начинается с буквы *а*, но и названия других цветов, в которых также содержится буква *а*. После ввода второй буквы (*к*) список сокращается до одного названия (*Маки*), содержащего введенное сочетание букв (*ак*). Виджет Autocomplete не проверяет правильность введенных данных, и пользователь может вводить в текстовом поле любые значения, а не только те, которые определены опцией `source`.

Заметьте, что никаких специальных мер по сортировке массива, указанного с помощью опции `source`, мы не предпринимали. За нас это автоматически сделала библиотека jQuery UI. Кроме того, как будет показано далее, возможные варианты продолжения ввода могут поступать из нескольких источников.

Совет. В приведенном примере документа элементы `input` и `label` помещены в элемент `div`, которому присвоен класс `ui-widget`. Тем самым обеспечивается совпадение CSS-свойств шрифтов, используемых в этих элементах и в раскрывающемся списке виджета Autocomplete. Более подробно об использовании CSS-классов библиотеки jQuery UI говорится в главе 34, тогда как небольшой демонстрационный пример приведен в главе 25.

Использование массива объектов в качестве источника данных

В качестве источника элементов списка автозаполнения вместо строк могут использоваться массивы объектов. Это дает возможность отделить ярлык, отображаемый в раскрывающемся списке, от значения, подставляемого в текстовое поле. Соответствующий демонстрационный пример приведен в листинге 19.2.

Листинг 19.2. Использование массива объектов в качестве источника данных для автозаполнения

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var flowers = [{label: "Астры (бордовые)", value:
                "Астры"}, {label: "Нарциссы (белые)", value:
                "Нарциссы"}, {label: "Розы (розовые)", value:
                "Розы"}, {label: "Пионы (розовые)", value:
                "Пионы"}]

            $('#acInput').autocomplete({
                source: flowers
            })
        });
    </script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Название цветов: </label>
            <input id="acInput"/>
        </div>
    </form>
</body>
</html>
```

Когда используется массив объектов, средство автозаполнения ищет свойства `label` и `value`. Свойство `label` используется для создания ярлыков, отображаемых в раскрывающемся списке, а свойство `value` — для вставки значений в поле ввода при выборе соответствующего ярлыка. В данном случае ярлыки содержат информацию об оттенках цветов, которая не включается в значения. Результат приведен на рис. 19.02.

Настройка виджета Autocomplete

Виджет Autocomplete поддерживает ряд настраиваемых свойств, позволяющих управлять различными аспектами его поведения. Перечень этих свойств приведен в табл. 19.2. Возможности настройки виджета Autocomplete описаны в следующих разделах.

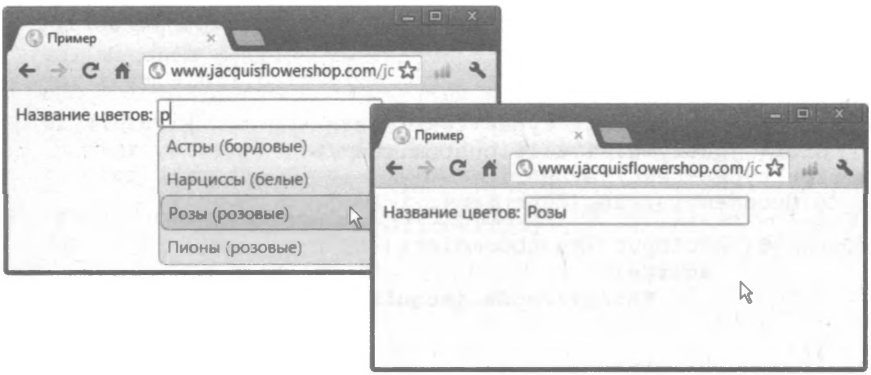


Рис. 19.2. Использование массива объектов для разделения ярлыков и значений

Таблица 19.2. Свойства виджета Autocomplete

Свойство	Описание
`appendTo`	Определяет элемент, к которому должен быть присоединен раскрывающийся список элементов автозаполнения. По умолчанию таковым является элемент `body`
`autoFocus`	Если эта опция равна `true`, то фокус автоматически устанавливается на первом элементе раскрывающегося списка, что позволяет выбирать его простым нажатием клавиши <Enter>. Значение по умолчанию — `false`
`delay`	Устанавливает длительность периода задержки (в миллисекундах) после нажатия клавиши, по истечении которого будет срабатывать автозаполнение. Значение по умолчанию — `300`
`disabled`	Установка значения `true` приводит к отключению средства автозаполнения. На базовый элемент `input` эта опция никакого влияния не оказывает. Значение по умолчанию — `false`
`minLength`	Определяет минимальное число символов, ввод которых будет инициировать отображение раскрывающегося списка элементов автозаполнения. Значение по умолчанию — `1`
`source`	Устанавливает источник данных для добавления в раскрывающийся список элементов автозаполнения. Значения по умолчанию для этого свойства не существует, и поэтому указание необходимого значения при вызове метода `autocomplete()` является обязательным

Использование дистанционного источника данных

Из всех настроек виджета Autocomplete наибольший интерес представляет опция `source`, поскольку она обеспечивает возможность выбора самых различных источников данных для включения в раскрывающийся список элементов автозаполнения. С ролью простых статических списков отлично справляются массивы JavaScript, которые использовались в предыдущем примере. В более сложных ситуациях необходимые данные могут поступать с сервера. Для этого потребуется лишь указать URL-адрес источника данных, как показано в листинге 19.3.

Листинг 19.3. Использование дистанционного источника данных

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
```

```
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#acInput').autocomplete({
                source:
                    "http://node.jacquisflowershop.com:9999/auto"
            })
        });
    </script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Название цветов: </label>
            <input id="acInput"/>
        </div>
    </form>
</body>
</html>
```

Как только средству автозаполнения понадобятся данные, jQuery UI выполнит HTTP-запрос по указанному URL-адресу, используя метод GET. Введенные к этому времени пользователем символы включаются в строку запроса с помощью ключевого слова `term`. Например, если пользователь введет букву *п*, то jQuery UI направит следующий запрос:

```
http://node.jacquisflowershop.com/auto?term=п
```

Если далее будет введена буква *и*, то запрос станет таким:

```
http://node.jacquisflowershop.com/auto?term=пи
```

Описанную методику удобно применять в тех случаях, когда имеется множество данных, которые нежелательно пересылать сразу все. Она также будет полезна, если список данных для автозаполнения динамически обновляется и вы хотите, чтобы пользователь имел доступ к самым свежим данным.

За получение значения `term` из строки запроса и возврат JSON-строки, представляющей массив элементов, отображаемых для пользователя, отвечает сервер. Простой серверный сценарий Node.js, пригодный для этих целей, приведен в листинге 19.4.

Листинг 19.4. Сценарий для сервера Node.js, поддерживающий дистанционное автозаполнение

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
    console.log("[200 OK] " + req.method + " to " + req.url);

    var flowers = ["Астры", "Нарциссы", "Розы", "Пионы",
        "Примулы", "Подснежники", "Маки", "Первоцветы", "Петунии",
        "Фиалки"];
```

```
    req.on('end', function() {
        var matches = [];
        var term = url.parse(req.url, true).query["term"];

        if (term) {
            var pattern = new RegExp("^" + term, "i");
            for (var i = 0; i < flowers.length; i++) {
                if (pattern.test(flowers[i])) {
                    matches.push(flowers[i]);
                }
            }
        } else {
            matches = flowers;
        }

        res.writeHead(200, "OK", {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*"});
        res.write(JSON.stringify(matches));
        res.end();
    });
}).listen(9999);
console.log("Ready on port 9999");
```

В этом сценарии используется тот же набор названий цветов, что и в предыдущем примере, и возвращаются значения, которые соответствуют поисковому выражению, определяемому ключевым словом `term` запроса, полученного от браузера. Я незначительно изменил процедуру поиска таким образом, чтобы возвращались лишь те названия цветов, которые начинаются с поискового термина. Так, если jQuery UI отправляет строку запроса

```
http://node.jacquisflowershop.com/auto?term=a
```

то сервер возвращает следующие данные JSON:

```
["Астры"
```

Поскольку поиск совпадений выполняется только в начале названий, список включает лишь один элемент, как показано на рис. 19.3.

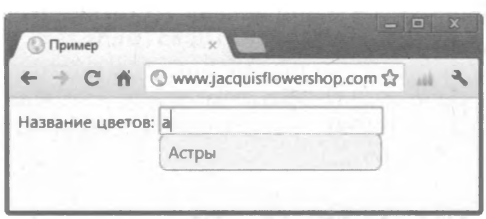


Рис. 19.3. Получение данных для автозаполнения с удаленного сервера

Эта методика действительно очень удобна, но может требовать отправки на сервер слишком большого количества запросов. Для рассмотренного примера это было не столь важно, поскольку выполнялся лишь простейший поиск, а мои сервер и браузер находятся в одной и той же сети. В то же время, в случае более сложных вариантов поиска в пределах глобальных сетей, когда возможны значительные задержки ответа, нагрузка на сервер может существенно возрасти.

Имеется возможность управлять частотой отправки запросов средством автозаполнения с помощью опций `miniLength` и `delay`. Опция `miniLength` позволяет указать минимальное количество символов, которые пользователь должен ввести, прежде чем jQuery UI направит соответствующий запрос серверу. Таким образом, к моменту отправки запроса пользователем уже будет введено несколько символов, и этой информации вам будет вполне достаточно, чтобы сузить область поиска.

С помощью опции `delay` можно указать время, которое должно пройти с момента ввода символа, прежде чем будет сделан запрос. Это позволяет избежать неоправданно частой отправки запросов при высокой скорости набора символов пользователем. Так, если пользователь введет буквы *п* и *и*, то будет направлен только один запрос, соответствующий комбинации *пи*, а не два мгновенно следующих один за другим отдельных запроса, соответствующих наборам букв *п* и *пи*. Совместно используя обе опции, можно уменьшить количество запросов и при этом по-прежнему предоставлять пользователю достаточный объем необходимой информации. Пример использования описанных опций настройки приведен в листинге 19.5.

Листинг 19.5. Уменьшение количества запросов за счет использования опций `miniLength` и `delay`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#acInput').autocomplete({
                source:
                    "http://node.jacquisflowershop.com:9999/auto",
                minLength: 3,
                delay: 1000
            })
        });
    </script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Название цветов: </label>
            <input id="acInput"/>
        </div>
    </form>
</body>
</html>
```

В этом примере запрос будет инициироваться лишь после того, как пользователь наберет три символа, и при условии, что в течение одной секунды после этого не будет введен ни один дополнительный символ.

Использование функции в качестве источника данных

Для создания источника элементов автозаполнения, в полной мере приспособленного к конкретным потребностям пользователей, можно использовать функцию. Эта функция также указывается с помощью опции `source` и вызывается всякий раз, когда средство автозаполнения должно отображать возможные варианты продолжения ввода для пользователей. Соответствующий пример приведен в листинге 19.6.

Листинг 19.6. Использование функции для генерации элементов автозаполнения

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript">
    </script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var flowers = ["Астры", "Нарциссы", "Розы", "Пионы",
                "Примулы", "Подснежники", "Маки", "Первоцветы",
                "Петунии", "Фиалки"];

            $('#acInput').autocomplete({
                source: function(request, response) {
                    var term = request.term;
                    var pattern = new RegExp("^" + term, "i");

                    var results = $.map(flowers, function(elem) {
                        if (pattern.test(elem)) {
                            return elem;
                        }
                    })
                    response(results);
                }
            })
        });
    </script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Название цветов: </label>
            <input id="acInput"/>
        </div>
    </form>
</body>
</html>
```

Мы передаем этой функции два аргумента. Один из них — это объект с единственным свойством `term`. Значением этого свойства является строка символов, введенных пользователем. Второй аргумент — это функция, которая вызывается, ко-

гда уже сгенерирован список элементов автозаполнения для отображения пользователю. Аргументом этой функции является массив строк или объектов.

В данном случае мы воспроизводим функциональность, которая использовалась на стороне сервера в предыдущем примере, и генерируем массив, включающий лишь те элементы автозаполнения, которые начинаются с заданного выражения.

Примечание. Для обработки содержимого массива используется вспомогательный метод jQuery `map()`, описанный в главе 33.

Далее результаты передаются обратно jQuery UI путем передачи массива функции `response()` в качестве аргумента.

```
response(results);
```

Данный способ обработки результатов не совсем обычен, но работает вполне удовлетворительно, и генерируемые элементы отображаются для пользователя, так что конечный результат получается тем же, что и в случае применения сервера Node.js.

Использование методов виджета Autocomplete

Виджет Autocomplete поддерживает ряд методов, которые можно использовать для управления процессом автозаполнения. Эти методы перечислены в табл. 19.3.

Таблица 19.3. Методы виджета Autocomplete

Метод	*Описание*
`autocomplete("close")`	Закрывает раскрывающийся список элементов автозаполнения
`autocomplete("destroy")`	Полностью удаляет функциональность автозаполнения из элемента `input`
`autocomplete("disable")`	Приостанавливает работу виджета Autocomplete для базового элемента
`autocomplete("enable")`	Возобновляет работу ранее приостановленного виджета Autocomplete
`autocomplete("option")`	Устанавливает значения одного или нескольких свойств
`autocomplete("search", значение)`	Запускает процедуру поиска элементов автозаполнения, соответствующих указанному значению. Если аргумент *значение* не предоставлен, используется содержимое элемента `input`

Специфичными для средства автозаполнения являются два метода — `search` и `close`. Их можно использовать для программного запуска и остановки процесса автозаполнения, как показано в листинге 19.7.

Листинг 19.7. Использование методов `search` и `close`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
```

```
    <style type="text/css">
        button {margin-bottom: 5px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            var flowers = ["Астры", "Нарциссы", "Розы", "Пионы",
                "Примулы", "Подснежники", "Маки", "Первоцветы",
                "Петунии", "Фиалки"];

            $('#acInput').autocomplete({
                source: flowers,
                minLength: 0,
                delay: 0
            })

            $('button').click(function(e) {
                e.preventDefault();
                switch (this.id) {
                    case "close":
                        $('#acInput').autocomplete("close");
                        break;
                    case "input":
                        $('#acInput').autocomplete("search");
                        break;
                    default:
                        $('#acInput')
                            .autocomplete("search", this.id);
                        break;
                }
            })
        });
    </script>
</head>
<body>
    <form>
        <button id="n">n</button>
        <button id="c">c</button>
        <button id="input">Содержимое ввода</button>
        <button id="close">Закрыть</button>
        <div class="ui-widget">
            <label for="acInput">Названия цветов: </label>
            <input id="acInput"/>
        </div>

    </form>
</body>
</html>
```

В этом примере мы добавили некоторые кнопки и использовали метод `click()` для настройки различных вызовов метода `autocomplete()`. При нажатии кнопок, обозначенных как П и С, вызывается метод `search`, которому выбранная буква передается в качестве поискового выражения. Это приводит к запуску средства автозаполнения с использованием выбранной буквы, независимо от содержимого текстового поля, как показано на рис. 19.4.

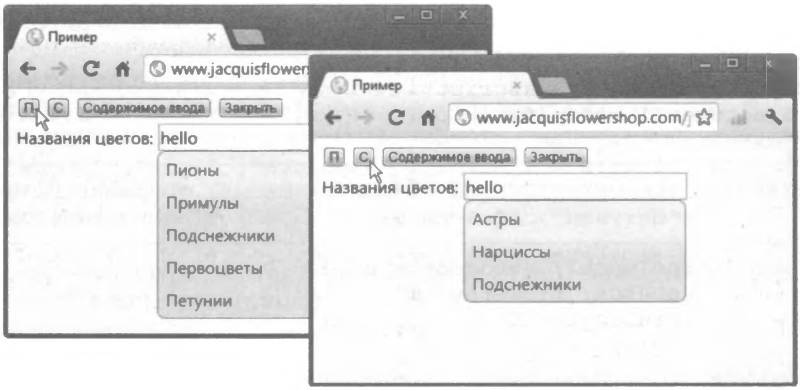


Рис. 19.4. Использование метода `search()` с поисковым выражением

Как показано на рисунке, в раскрывающемся списке отображаются элементы, которые содержат букву, указанную на нажатой кнопке, несмотря на то, что в текстовом поле введено слово `hello`.

Кнопка **Содержимое ввода** запускает средство автозаполнения с использованием тех символов, которые содержатся в текстовом поле. При настройке средства автозаполнения свойству `miniLength` было присвоено значение `0`, в результате чего после щелчка на кнопке **Содержимое ввода** в списке ввода отображаются сразу все элементы автозаполнения, как показано на рис. 19.5. Если пользователь введет какие-либо символы в поле ввода, то отобразятся лишь элементы, содержащие введенную строку.

Использование событий виджета Autocomplete

С виджетом Autocomplete связан ряд событий, которые перечислены в табл. 19.4.

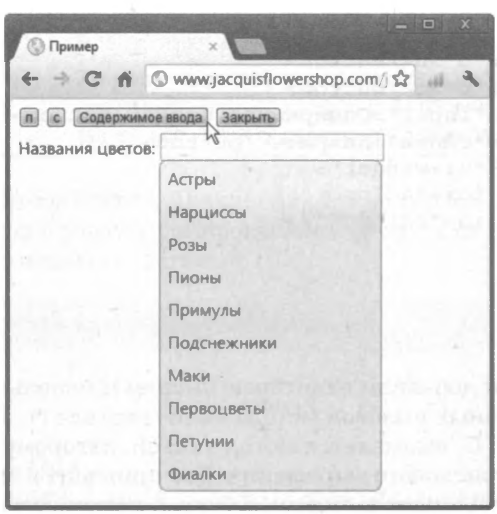


Рис. 19.5. Поиск с использованием содержимого поля ввода

Таблица 19.4. События виджета Autocomplete

Событие	*Описание*
`change`	Происходит, когда фокус ввода выходит за пределы текстового поля после изменения содержащегося в нем значения
`close`	Происходит при закрытии раскрывающегося списка элементов автозаполнения
`create`	Происходит при инициализации экземпляра виджета Autocomplete для данного элемента
`focus`	Происходит, когда элемент автозаполнения получает фокус в раскрывающемся списке
`open`	Происходит при открытии раскрывающегося списка элементов автозаполнения
`search`	Происходит перед генерацией раскрывающегося списка элементов автозаполнения или перед поиском в нем подходящих элементов
`select`	Происходит при выборе элемента списка автозаполнения

Получение сведений о выбранном элементе

В jQuery UI можно получить дополнительную информацию о событии с помощью второго аргумента функции-обработчика, обычно обозначаемого как `ui`. В случае событий `change`, `focus` и `select` jQuery UI наделяет объект `ui` свойством `item`, возвращающим объект, который описывает выбранный или получивший фокус элемент раскрывающегося списка. Пример, в котором демонстрируется использование этой возможности для получения информации о выбранном элементе, приведен в листинге 19.8.

Листинг 19.8. Использование объекта `ui` в обработчиках событий

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var flowers = ["Астры", "Нарциссы", "Розы", "Пионы",
                "Примулы", "Подснежники", "Маки", "Первоцветы",
                "Петунии", "Фиалки"];

            $('#acInput').autocomplete({
                source: flowers,
                focus: displayItem,
                select: displayItem,
                change: displayItem
            })

            function displayItem(event, ui) {
                $('#itemLabel').text(ui.item.label)
            }
        });
```

```
    </script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Названия цветов: </label>
            <input id="acInput"/>
            Ярлык элемента: <span id="itemLabel"></span>
        </div>
    </form>
</body>
</html>
```

В этом примере добавлен элемент `span`, который используется для отображения свойства `label` выбранного объекта. Объекты со свойствами `label` и `value` создаются jQuery UI даже тогда, когда вы задаете в опции `source` простой массив строк, поэтому одно из этих свойств объекта `ui.item` вам всегда приходится считывать. В данном примере для событий `focus`, `select` и `change` используется одна и та же функция, отображающая ярлык элемента. Результат представлен на рис. 19.6.

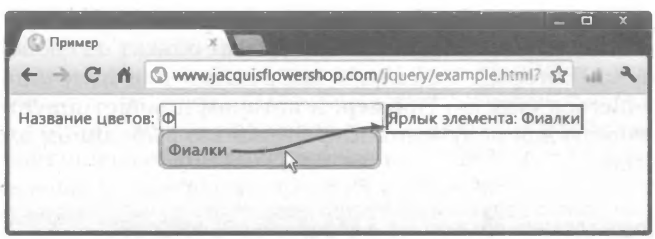


Рис. 19.6. Получение сведений о выбранном элементе

Замена действия по умолчанию для события `select`

Для события `select` предусмотрено действие по умолчанию, которое заключается в замене содержимого элемента `input` содержимым свойства `value` элемента, выбранного в раскрывающемся списке. Это как раз то, что необходимо в большинстве случаев, но можно использовать данное событие и для того, чтобы либо дополнить действие по умолчанию, либо полностью отменить его и выполнить какие-то другие действия, которые вам нужны. Пример дополнения действия по умолчанию установкой значения связанного текстового поля приведен в листинге 19.9.

Листинг 19.9. Замена действия по умолчанию для события `select`

```
<!DOCTYPE html>
<html>
<head>

    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
```

```
            var flowers = ["Астры", "Нарциссы", "Розы"];

            var skus = {Астры: 100, Нарциссы: 101, Розы: 102};

            $('#acInput').autocomplete({
                source: flowers,
                select: function(event, ui) {
                    $('#sku').val(skus[ui.item.value]);
                }
            })
        });
    </script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Названия цветов: </label>
            <input id="acInput"/>
            <label for="sku">Секция склада: </label>
            <input id="sku"/>
        </div>
    </form>
</body>
</html>
```

Когда происходит событие `select`, функция-обработчик использует аргумент `ui` для получения значения выбранного элемента и установки значения связанного поля, в данном случае — поля Секция склада, которое получается из объекта `skus`. Тем самым мы можем помогать пользователям, предоставляя значения по умолчанию для других полей на основе первоначального выбора. Это может пригодиться во множестве ситуаций, особенно при выборе таких элементов, как адреса доставки. Результаты представлены на рис. 19.7, однако с примерами подобного рода лучше всего знакомиться, выполняя их непосредственно в браузере. HTML-код этого документа, а также всех других примеров, приводимых в книге, можно скачать на сайте книги (см. главу 1).

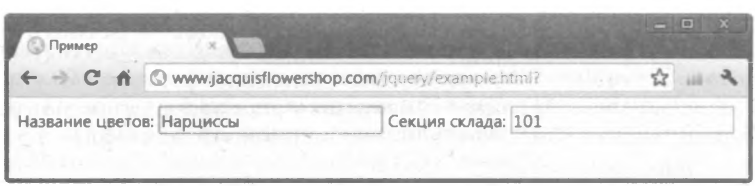


Рис. 19.7. Использование события `select` для заполнения другого поля

Использование виджета Accordion

Виджет Accordion позволяет представить набор панелей, имеющих информационное содержимое, таким образом, чтобы в каждый момент времени отображалась лишь одна из них. При выборе пользователем другой панели отображавшаяся до этого панель закрывается, а новая открывается, что создает эффект, напоминающий растяжение мехов гармошки.

Панели виджета Accordion отлично подходят для отображения содержимого в ситуациях, когда оно представляется в виде нескольких независимых частей, и вы

не хотите перегружать пользователя, отображая их все одновременно. В идеальном случае отдельные панели содержимого объединяет какая-либо общая тема, основное содержание отдельных разделов которой может быть выражено простыми заголовками.

Создание виджета Accordion

Как вы, наверное, уже догадались, для создания виджета Accordion jQuery UI используется метод `accordion()`. Пример создания такого виджета представлен в листинге 19.10.

Листинг 19.10. Создание виджета Accordion

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #accordion {margin: 5px}
        .dcell img {height: 60px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"},
                        {"name":"Примулы","product":"primula"},
                        {"name":"Подснежники",
                            "product":"snowdrop"},
                        {"name":"Гвоздики","product":"carnation"},
                        {"name":"Лилии","product":"lily"},
                        {"name":"Орхидеи","product":"orchid"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 3).appendTo("#row1");
            elems.slice(3, 6).appendTo("#row2");
            elems.slice(6).appendTo("#row3");

            $('#accordion').accordion();

            $('button').button();
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
```

```
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="accordion">
            <h2><a href="#">Ряд 1</a></h2>
            <div id="row1"></div>
            <h2><a href="#">Ряд 2</a></h2>
            <div id="row2"></div>
            <h2><a href="#">Ряд 3</a></h2>
            <div id="row3"></div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button></div>
    </form>
</body>
</html>
```

Наиболее важной частью этого примера является содержимое контейнера `div` с идентификатором `accordion`.

```
<div id="accordion">

    <h2><a href="#">Ряд 1</a></h2>
    <div id="row1"></div>

    <h2><a href="#">Ряд 2</a></h2>
    <div id="row2"></div>

    <h2><a href="#">Ряд 3</a></h2>
    <div id="row3"></div>

</div>
```

Здесь форматирование HTML-разметки было незначительно изменено, чтобы сделать структуру элементов более очевидной. Элемент `div` верхнего уровня — это базовый элемент, для которого вызывается метод `accordion()`. После вызова этого метода jQuery UI просматривает содержимое базового элемента в поиске заголовков (элементы от `h1` до `h6`) и разбивает содержимое на отдельные разделы таким образом, чтобы с каждым заголовком был связан следующий за ним элемент. В данном случае в качестве заголовков используются элементы `h2`, за каждым из которых следует элемент `div`. Для наполнения элементов `div` информацией о продукции, предлагаемой в цветочном магазине, в сценарии используется подключаемый модуль шаблонов.

Обратите внимание на элементы `a`, добавленные в каждый элемент `h2`. С помощью этих элементов мы определяем заголовки для панелей содержимого. Как именно jQuery UI выполняет преобразование элемента `div` верхнего уровня и его содержимого, показано на рис. 19.8.

Совет. Присвоение атрибуту `href` значения `#` — это общепринятый способ идентификации элементов в тех случаях, когда с ними предполагают работать только средствами JavaScript. Этот подход использован здесь исключительно ради упрощения примера, но в целом я рекомендую использовать динамическую вставку элементов `a` с помощью jQuery, чтобы не создавать помех для тех пользователей, в браузерах которых выполнение JavaScript-сценариев запрещено.

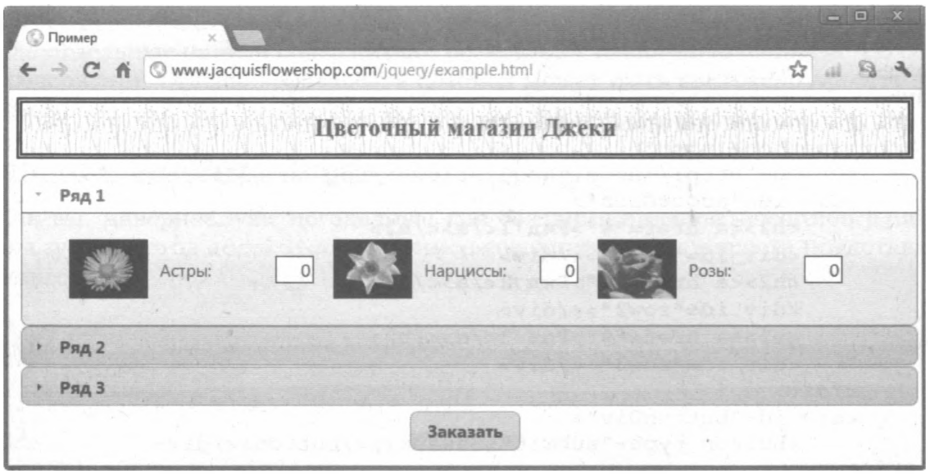


Рис. 19.8. Виджет Accordion jQuery UI

Во вновь созданном виджете отображается лишь содержимое первой панели, в то время как остальные панели закрыты. Содержимое элементов a используется в качестве заголовков панелей, и щелчок на каком-либо заголовке приводит к закрытию текущей панели и открытию панели, заголовок которой был выбран (этот переход выполняется с применением эффекта анимации, который трудно передать с помощью снимков экрана). Результат выполнения щелчков на заголовках представлен на рис. 19.9.

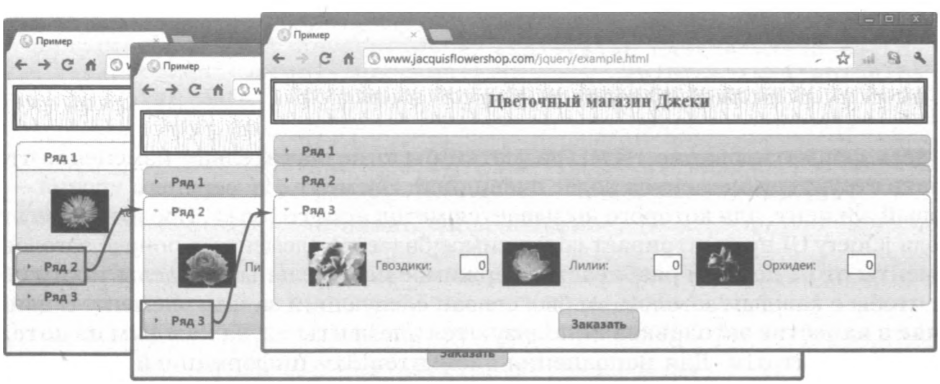


Рис. 19.9. Переходы между панелями содержимого

Настройка виджета Accordion

Виджет Accordion поддерживает ряд настраиваемых свойств, позволяющих управлять различными аспектами его поведения. Эти свойства приведены в табл. 19.5. Возможности настройки виджета Autocomplete с помощью этих опций описаны в следующих разделах.

Таблица 19.5. Свойства виджета Accordion

Свойство	*Описание*
`active`	Позволяет определить, какая панель отображается или должна отображаться. По умолчанию первоначально отображается первая панель
`animated`	Определяет тип анимации, используемой при смене панелей содержимого. Значение по умолчанию — `slide`. Подробно возможности анимации jQuery UI описаны в главе 34
`autoHeight`	Если эта опция равна `true`, то для всех панелей устанавливается одинаковая высота, соответствующая панели с наибольшим по размеру содержимым. Значение по умолчанию — `true`
`clearStyle`	Если эта опция равна `true`, то по завершении анимационного перехода значения высоты всех панелей сбрасываются и заново пересчитываются для учета возможного изменения высоты панели с максимальным размером содержимого. Значение по умолчанию — `false`
`collapsible`	Если эта опция равна `true`, то щелчок на открытой панели приводит к ее закрытию, в результате чего все панели оказываются закрытыми. Значение по умолчанию — `false`
`disabled`	Если эта опция равна `true`, то виджет отключен. Значение по умолчанию — `false`
`event`	Определяет для элемента заголовка событие, при наступлении которого происходит переключение панелей. Значение по умолчанию — `click`
`fillSpace`	Если эта опция равна `true`, то виджет будет занимать все пространство своего родительского элемента по высоте. Значение по умолчанию — `false`, при котором высота виджета определяется высотой панелей содержимого
`header`	Определяет элементы, используемые в качестве заголовков
`icons`	Определяет значки, используемые в виджете Accordion

Настройка высоты виджета Accordion

Высоту виджета Accordion можно регулировать различными способами, в которых исходят либо из высоты содержимого панелей, либо из высоты родительского элемента. Чаще всего можно ничего не менять и оставить значение по умолчанию, определяемое опцией `autoHeight`. Эта опция, значением которой по умолчанию является `true`, устанавливает для всех панелей одинаковую высоту (совпадающую с высотой максимальной по размеру содержимого панели) и на основании этого определяет высоту всего виджета.

Именно такой подход был использован в предыдущем примере, однако им необходимо пользоваться с осторожностью в тех случаях, когда содержимое включает изображения, особенно если элементы `img` добавляются в документ средствами jQuery. Проблема в том, что метод `accordion()` может быть вызван еще до того, как будут загружены все изображения, в результате чего информация о высоте панелей содержимого, полученная jQuery UI от браузера, окажется ложной. В моем браузере высота элементов `div` содержимого составляла 55 пикселей до загрузки изображений и 79 пикселей после их загрузки. О возникновении проблемы такого рода можно судить по неожиданному появлению полос прокрутки при отображении содержимого, как показано на рис. 19.10.

Поскольку jQuery UI не отслеживает изменение высоты панелей содержимого при загрузке изображений, содержимое отображается некорректно. Чтобы справиться с этой проблемой, необходимо предоставить информацию о том, какой будет высота панелей после загрузки всех внешних ресурсов. Это можно сделать

разными способами. В данном случае я достиг требуемого результата, установив подходящее значение CSS-свойства `height` элементов `img` в элементе `style`.

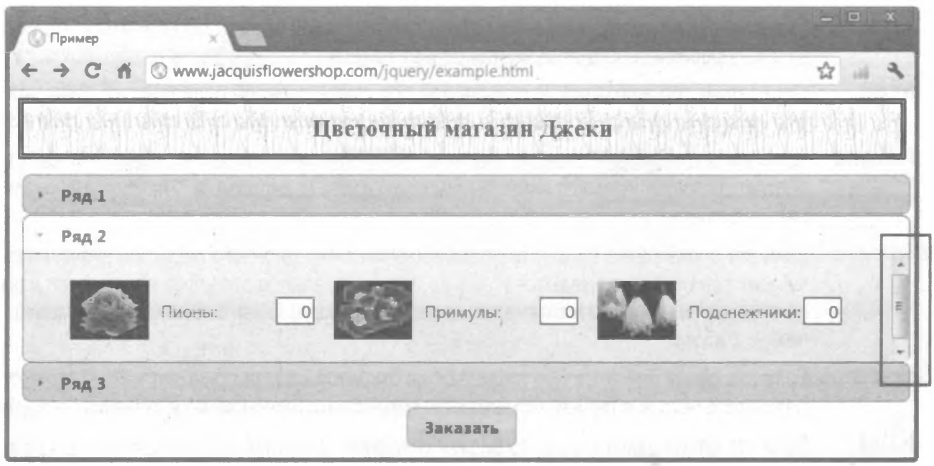


Рис. 19.10. Проблемы, обусловленные использованием недостоверной информации о высоте элементов

```
...
<style type="text/css">
    #accordion {margin: 5px}
    .dcell img {height: 60px}
</style>
...
```

Продолжим наш пример. Используя значение по умолчанию опции `autoHeight`, равное `true`, можно установить одинаковую высоту для всех панелей содержимого, что будет выглядеть не очень привлекательно при больших отличиях в размерах содержимого различных панелей. В листинге 19.11 приведен сценарий, в котором способ вставки элементов, содержащих информацию о продуктах, несколько изменен.

Листинг 19.11. Пример виджета Accordion с большими различиями высоты содержимого разных панелей

```
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"},
                        {"name":"Примулы","product":"primula"},
                        {"name":"Подснежники",
                            "product":"snowdrop"},
                        {"name":"Гвоздики","product":"carnation"},
                        {"name":"Лилии","product":"lily"},
                        {"name":"Орхидеи","product":"orchid"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 3).appendTo("#row1");
```

```
            elems.slice(3, 6).appendTo("#row2");
            elems.slice(6).appendTo("#row3");

            $('<h2><a href=#>Все</a></h2><div id=row0></div>')
                .prependTo('#accordion')
                .filter('div').append($('#row1, #row2, #row3')
                .clone())

            $('#accordion').accordion();

            $('button').button();
        });
    </script>
...
```

В этом примере для создания панели содержимого, высота которой значительно превышает высоту остальных панелей, имеющиеся элементы содержимого `div` клонируются с помощью jQuery и помещаются в новую панель, в результате чего создается панель, в которой отображаются все продукты. Высота этой панели в раскрытом состоянии в три раза превышает высоту любой другой панели, что приводит к образованию пустого пространства при отображении панелей с содержимым небольшого размера. Все это можно увидеть на рис. 19.11.

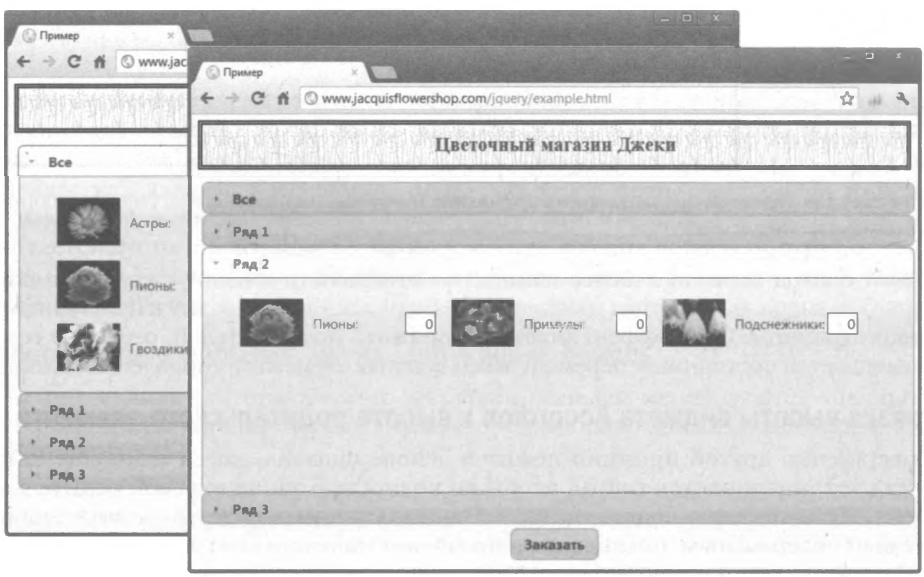


Рис. 19.11. Эффект использования опции `autoHeight` в случае больших различий в высоте элементов содержимого

Если большие размеры пустого пространства в пределах панелей вас не устраивают, можете установить для опции `autoHeight` значение `false`, как показано в листинге 19.12.

Листинг 19.12. Отключение средства `autoHeight`

```
...
$('#accordion').accordion({
```

```
    autoHeight: false});
})
...
```

Теперь высота виджета будет динамически изменяться при смене панелей. Результат представлен на рис. 19.12.

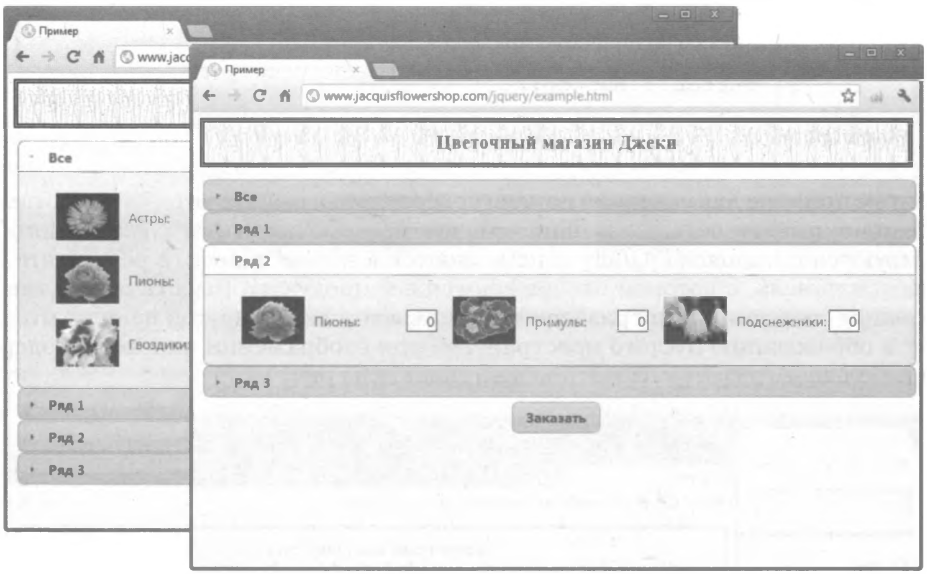


Рис. 19.12. Изменение размера гармошки в соответствии с высотой отображаемого содержимого

Такой подход позволяет более компактно отображать панели с содержимым небольшого размера, но при этом изменение размера панели приводит к изменению компоновки страницы. Этот эффект может раздражать пользователей, особенно если он сопровождается постоянным перемещением важных элементов управления на экране.

Привязка высоты виджета Accordion к высоте родительского элемента

Совершенно другой принцип лежит в основе подхода, когда значение высоты виджета устанавливается таким, чтобы он полностью заполнял свой родительский элемент. Я считаю этот подход наиболее полезным при работе с динамически генерируемым содержимым, поскольку он позволяет надежно контролировать размеры и не сопровождается изменением компоновки страницы. Для этого следует использовать опцию `fillSpace`, как показано в листинге 19.13.

Листинг 19.13. Установка высоты виджета Accordion на основании высоты родительского элемента

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"},
```

```
                    {"name":"Примулы","product":"primula"},
                    {"name":"Подснежники",
                        "product":"snowdrop"},
                    {"name":"Гвоздики","product":"carnation"},
                    {"name":"Лилии","product":"lily"},
                    {"name":"Орхидеи","product":"orchid"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 3).appendTo("#row1");
            elems.slice(3, 6).appendTo("#row2");
            elems.slice(6).appendTo("#row3");

            $('<h2><a href=#>Все</a></h2><div id=row0></div>')
                .prependTo('#accordion')
                .filter('div')
                .append($('#row1, #row2, #row3').clone())

            $('#accordion')
                .wrap('<div style="height:300px"></div>');

            $('#accordion').accordion({
                fillSpace: true);

            $('button').button();
        });
    </script>
...
```

В этом примере элемент с идентификатором `accordion` помещается в новый родительский элемент `div` с фиксированным размером 300 пикселей. При вызове метода `accordion()` свойству `fillSpace` присваивается значение `true`. Если размер родительского элемента меньше размера элемента, представляющего панель содержимого, к панели добавляется полоса прокрутки, в противном случае к элементу добавляется пустое пространство. Пример использования полосы прокрутки показан на рис. 19.13. Ее появление объясняется тем, что высота элемента, отображающего все виды цветов, превышает высоту родительского элемента, равную 300 px.

Изменение типа события, активизирующего панель

По умолчанию для открытия и закрытия панелей используются щелчки мышью. Поведением панелей можно управлять с помощью опции `event`, как показано в листинге 19.14.

Листинг 19.14. Использование опции `event`

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"},
                        {"name":"Примулы","product":"primula"},
                        {"name":"Подснежники",
                            "product":"snowdrop"},
                        {"name":"Гвоздики","product":"carnation"},
                        {"name":"Лилии","product":"lily"},
                        {"name":"Орхидеи","product":"orchid"}];
```

```
        var elems = $('#flowerTmpl').tmpl(data);
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3, 6).appendTo("#row2");
        elems.slice(6).appendTo("#row3");

        $('#accordion').accordion({
            event: "mouseover"
        })

        $('button').button();
    });
</script>
```

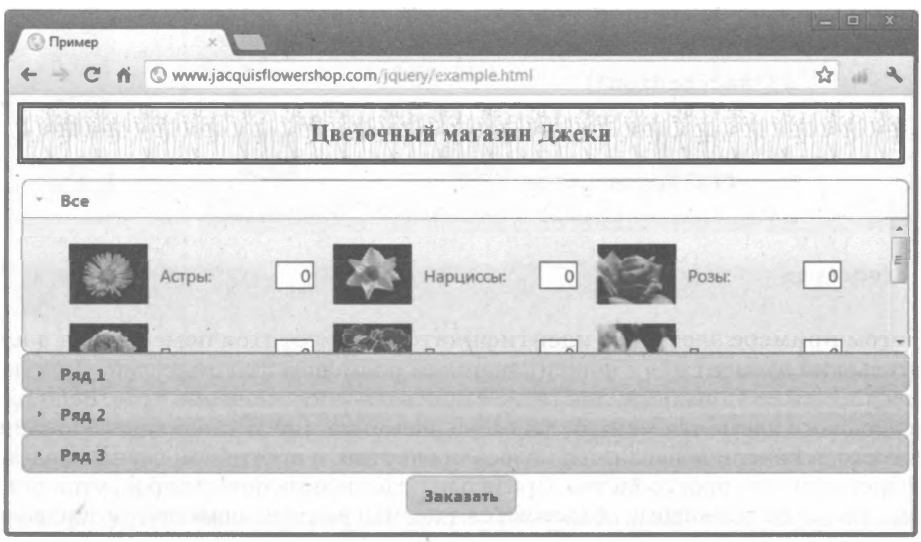


Рис. 19.13. Виджет с панелью, содержимое которой заполняет родительский элемент

В этом примере мы указали с помощью опции `event`, что панели должны открываться в ответ на событие `mouseover` (описано в главе 9). В результате этого открытие панели будет происходить при наведении на нее указателя мыши. Я не могу продемонстрировать этот эффект с помощью снимков экрана и поэтому рекомендую выполнить код примера в браузере. Описанный эффект выглядит привлекательно, однако применять его следует с осторожностью. Обычно пользователи быстро привыкают к тому, что для взаимодействия с элементами управления на странице используются щелчки, и поэтому "самопроизвольное" открытие панелей поначалу может их смущать.

Выбор активного заголовка

По умолчанию при первоначальном появлении виджета Accordion на экране открывается первая панель. Такое поведение виджета можно изменить с помощью опции `active`. Значениями этой опции могут быть строка селектора, объект `jQuery`, объект `HTMLElement` или число. В первых двух случаях используется первый подходящий элемент, тогда как числовое значение используется в качестве индекса, отсчитываемого от нуля. Пример использования строки селектора представлен в листинге 19.15.

Листинг 19.15. Использование опции `active`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #accordion {margin: 5px}
        .dcell img {height: 60px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"},
                        {"name":"Примулы","product":"primula"},
                        {"name":"Подснежники",
                            "product":"snowdrop"},
                        {"name":"Гвоздики","product":"carnation"},
                        {"name":"Лилии","product":"lily"},
                        {"name":"Орхидеи","product":"orchid"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 3).appendTo("#row1");
            elems.slice(3, 6).appendTo("#row2");
            elems.slice(6).appendTo("#row3");

            $('#accordion').accordion({
                active: "#row2header"
            });

            $('button').button();
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="accordion">
            <h2><a href="#">Ряд 1</a></h2>
```

```
            <div id="row1"></div>
            <h2 id="row2header"><a href="#">Ряд 2</a></h2>
            <div id="row2"></div>
            <h2><a href="#">Ряд 3</a></h2>
            <div id="row3"></div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button></div>
    </form>
</body>
</html>
```

Используя опцию `active`, важно помнить, что селектор применяется к элементам заголовков. В данном примере это элементы `h2`, а не `div` или `a`. Я добавил атрибут `id` в один из элементов `h2` и использовал его в качестве строки селектора. В результате виджет первоначально раскрывается на второй панели, как показано на рис. 19.14.

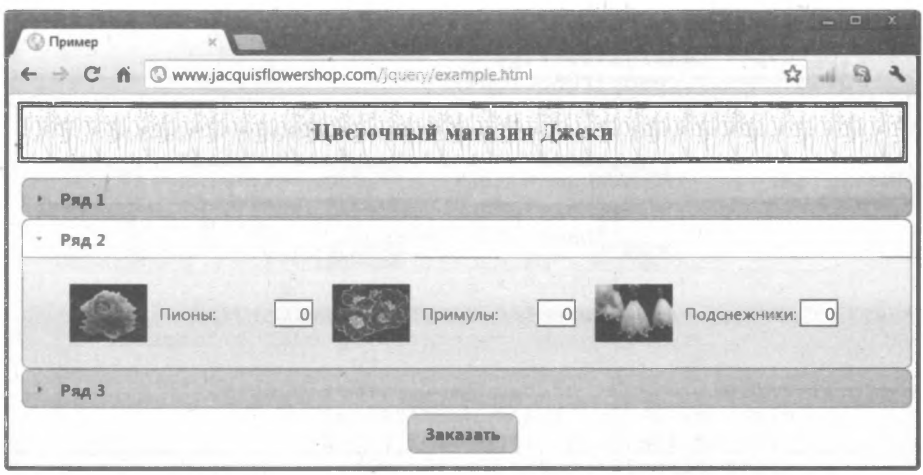


Рис. 19.14. Результат предварительного выбора открывающейся панели

В результате установки значения `false` для свойства `active` все панели виджета при его первоначальном отображении окажутся закрытыми. При этом значение свойства `collapsible` должно быть установлено равным `true`. Это приведет к отключению политики, используемой по умолчанию, в соответствии с которой в любой момент должна быть открыта хотя бы одна панель виджета. Пример использования этих настроек приведен в листинге 19.16.

Листинг 19.16. Отключение первоначальной активизации панелей

```
...
$('#accordion').accordion({
    active: false,
    collapsible: true
    });
```

Результат применения указанных настроек показан на рис. 19.15.

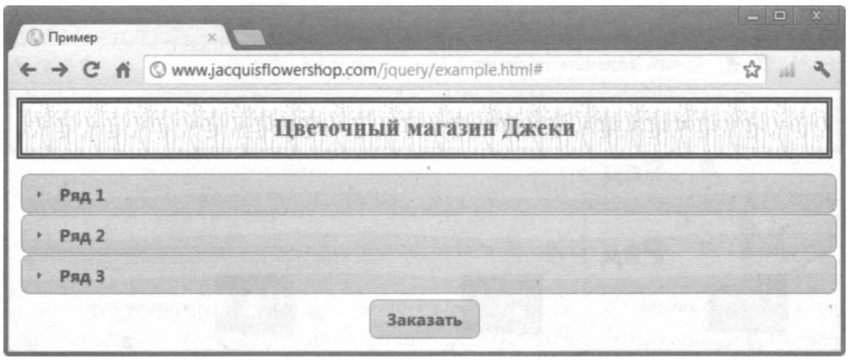


Рис. 19.15. Виджет Accordion с первоначально закрытыми панелями

При этом виджет будет работать, как обычно, за исключением того, что первоначально ни одна из панелей не будет активной, и пользователь, если это потребуется, сможет в любой момент закрыть все панели. Этот прием удобно использовать в условиях ограниченного экранного пространства, особенно если содержимое панелей не представляет особого интереса для пользователя.

Изменение значков виджета Accordion

Опция `icons` позволяет изменить значки, используемые в заголовках панелей виджета. Соответствующий пример представлен в листинге 19.17.

Листинг 19.17. Изменение значков, используемых в виджете Accordion

```
...
$('#accordion').accordion({
    collapsible: true,
    icons: {
        header: "ui-icon-zoomin",
        headerSelected: "ui-icon-zoomout"
    }
});
...
```

Значение опции `icons` задается в виде объекта, имеющего свойства `header` и `headerSelected`. Первое свойство указывает, какой значок следует использовать для закрытой панели, а второе — для открытой. Обычно я использую опцию `icons` в сочетании с опцией `collapsible`, поскольку использование значков как дополнительного средства, указывающего на возможность выполнения тех или иных действий, повышает комфортность работы пользователя. Как эти значки выглядят в заголовках панелей, показано на рис. 19.16.

Использование методов виджета Accordion

Для виджета Accordion определен ряд методов. Большинство из них — это стандартные методы, общие для всех виджетов jQuery UI, но два из них обеспечивают возможность управления данным виджетом из кода программы. Список доступных методов приведен в табл. 19.6.

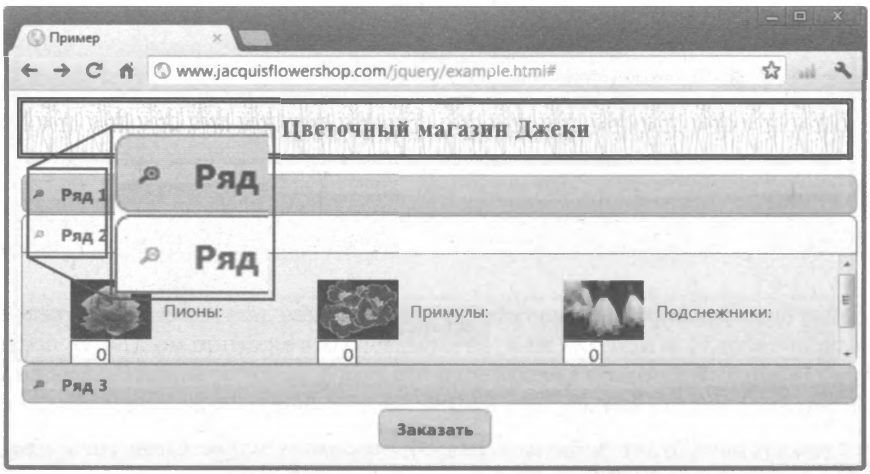


Рис. 19.16. Применение настраиваемых значков в заголовках панелей

Таблица 19.6. Методы виджета Accordion

Метод	*Описание*
`accordion("destroy")`	Полностью удаляет функциональность виджета Accordion из элемента `input`
`accordion("disable")`	Приостанавливает работу виджета Accordion для данного элемента
`accordion("enable")`	Возобновляет работу ранее приостановленного виджета Accordion
`accordion("option")`	Устанавливает значения одной или нескольких опций
`accordion("activate", index)`	Открывает панель, соответствующую указанному индексу (отсчет индексов ведется с нуля)
`accordion("resize")`	При включенной опции `fillSpace` разрешает изменение размера виджета в соответствии с изменениями размера родительского элемента

Метод `resize` лучше всего использовать совместно с элементами, обладающими функциональностью Draggable, которая описана в главе 24. Поэтому в данной главе нам остается рассмотреть лишь метод `activate`, поскольку все остальные методы являются общими для всех виджетов jQuery UI, и о том, как они работают, уже рассказывалось в главе 18. Пример использования метода `activate` для программного управления виджетом Accordion приведен в листинге 19.18.

Листинг 19.18. Управление виджетом Accordion с помощью метода `activate`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = [{"name":"Астры","product":"astor"},
                    {"name":"Нарциссы","product":"daffodil"},
                    {"name":"Розы","product":"rose"},
                    {"name":"Пионы","product":"peony"},
                    {"name":"Примулы","product":"primula"},
                    {"name":"Подснежники",
```

```
                    "product":"snowdrop"},
                {"name":"Гвоздики","product":"carnation"},
                {"name":"Лилии","product":"lily"},
                {"name":"Орхидеи","product":"orchid"}];

        var elems = $('#flowerTmpl').tmpl(data);
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3, 6).appendTo("#row2");
        elems.slice(6).appendTo("#row3");

        $('#accordion').accordion({
            active: false,
            collapsible: true
        });

        $('button').hide();
        var ids = ["2", "1", "0", "Скрыть"];
        for (var i = 0; i < ids.length; i++) {
            $('<button id=' + ids[i] + '>' + ids[i] + '</button>')
                .insertAfter('h1')
}

        $('button').button().click(function(e) {
            if (this.id == "Скрыть") {
                $('#accordion').accordion("activate", false);
            } else {
                $('#accordion')
                    .accordion("activate", Number(this.id));
            }
        });
    });
</script>
...
```

В этом сценарии добавлены кнопки, соответствующие индексам панелей, а также кнопка Скрыть, щелчок на которой приводит к закрытию всех панелей. Результат можно видеть на рис. 19.17.

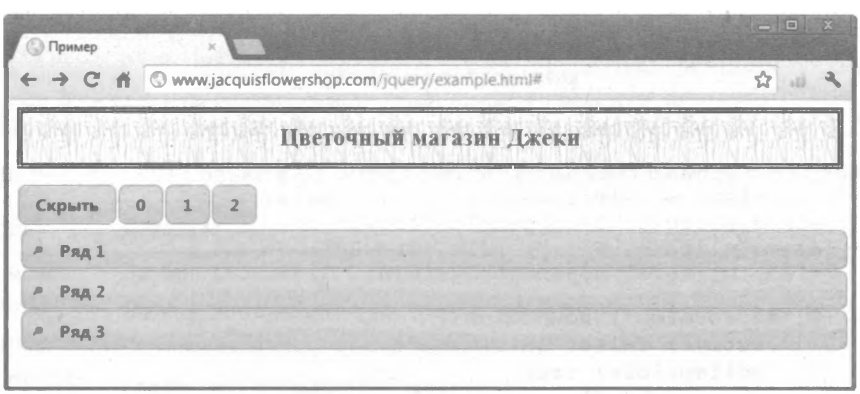


Рис. 19.17. Добавление кнопок для управления активизацией панелей

Щелчок на пронумерованной кнопке активизирует соответствующую панель. Это достигается с помощью метода `activate`.

```
$('#accordion').accordion("activate", Number(this.id));
```

С помощью второго аргумента методу передается атрибут `id` соответствующей кнопки. Чтобы отключить все панели, для этого аргумента следует указать значение `false`.

```
$('#accordion').accordion("activate", false);
```

Значение `false` работает только в том случае, если свойство `collapsible` равно `true`, в противном случае это значение игнорируется.

Использование событий виджета Accordion

Виджет Accordion поддерживает три события, которые перечислены в табл. 19.7.

Таблица 19.7. События виджета Accordion

Событие	*Описание*
`create`	Происходит при создании виджета
`change`	Происходит при переключении панелей виджета
`changestart`	Происходит в начале процесса переключения панелей виджета

События `changestart` и `change` можно использовать для отслеживания переключения активных панелей, как показано в листинге 19.19.

Листинг 19.19. Использование события `change`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = [{"name":"Астры","product":"astor"},
                    {"name":"Нарциссы","product":"daffodil"},
                    {"name":"Розы","product":"rose"},
                    {"name":"Пионы","product":"peony"},
                    {"name":"Примулы","product":"primula"},
                    {"name":"Подснежники",
                        "product":"snowdrop"},
                    {"name":"Гвоздики","product":"carnation"},
                    {"name":"Лилии","product":"lily"},
                    {"name":"Орхидеи","product":"orchid"}];

        var elems = $('#flowerTmpl').tmpl(data);
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3, 6).appendTo("#row2");
        elems.slice(6).appendTo("#row3");

        $('#accordion').accordion({
            active: false,
            collapsible: true,
            change: handleAccordionChange
        })

        function handleAccordionChange(event, ui) {
```

```
            var contentElems = $('#accordion').children('div');

            if (ui.oldContent.length) {
                var oldIndex = contentElems.index(ui.oldContent);
                $('button[id=' + oldIndex + ']').button("enable");
            } else {
                $('button[id=None]').button("enable");
            }
            if (ui.newContent.length) {
                var newIndex = contentElems.index(ui.newContent);
                $('button[id=' + newIndex + ']')
                    .button("disable");
            } else {
                $('button[id=None]').button("disable");
        }

        $('button').hide();
        var ids = ["2", "1", "0", "Скрыть"];
        for (var i = 0; i < ids.length; i++) {
            $('<button id=' + ids[i] + '>' + ids[i] + '</button>')
                .insertAfter('h1')
}

        $('button').button().click(function(e) {
            if (this.id == "Скрыть") {
                $('#accordion').accordion("activate", false);
            } else {
                $('#accordion')
                    .accordion("activate", Number(this.id));
            }
        });
    });
</script>
...
```

В этом сценарии для реагирования на смену активной панели используется событие change. Мы включаем и отключаем динамически добавленные кнопки таким образом, чтобы кнопка, соответствующая активной панели, была отключена. Если активные панели отсутствуют, то кнопка Скрыть отключается. Когда вы используете событие change или changestart, jQuery UI передает вам информацию об активных панелях посредством дополнительного аргумента функции-обработчика точно так же, как и в случае виджета Autocomplete. Этот дополнительный аргумент, обычно обозначаемый как ui, имеет свойства, перечисленные в табл. 19.8.

Таблица 19.8. Свойства объекта ui, используемого событиями change и changestart

Свойство	*Описание*
newHeader	Элемент заголовка для вновь активизированной панели
oldHeader	Элемент заголовка для предыдущей активной панели
newContent	Содержимое вновь активизированной панели
oldContent	Содержимое предыдущей активной панели

В этом примере мы определяем номера позиций текущей и предыдущей активных панелей, используя свойства newContent и oldContent в сочетании с методом

index(). Полученные значения соответствуют атрибутам id кнопок, с помощью которых можно открывать и закрывать нужные панели. Изменение состояния кнопки приводит к изменению ее внешнего вида, как показано на рис. 19.18.

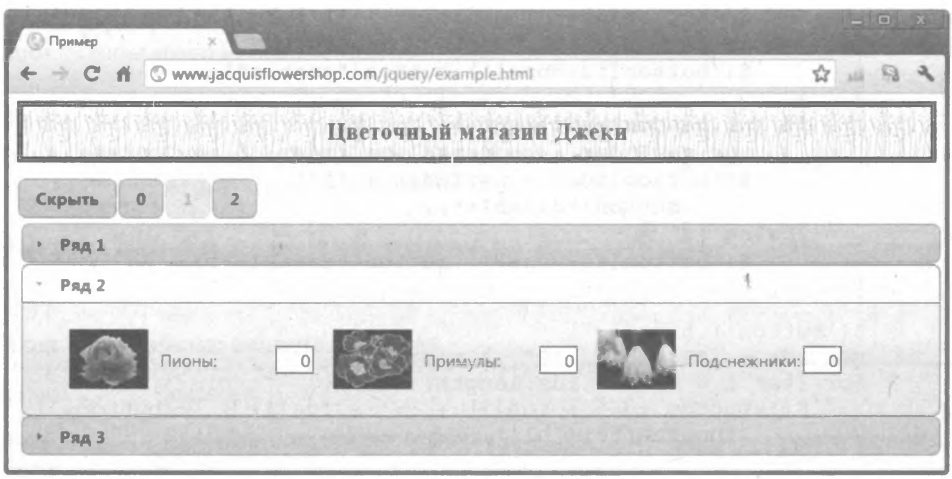


Рис. 19.18. Использование событий виджета Accordion для управления состоянием кнопок

Резюме

В этой главе вы познакомились с виджетами jQuery UI Autocomplete и Accordion. Они организованы аналогично виджетам, описанным в главе 18, но предлагают более богатую функциональность и более широкий ряд настраиваемых свойств, позволяющих конфигурировать виджеты таким образом, чтобы они как можно лучше вписывались в модель веб-приложения.

ГЛАВА 20

Использование виджета Tabs

Эта глава посвящена виджету Tabs. Внешне он напоминает виджет Accordion, рассмотренный в главе 18, но обладает более развитой функциональностью и предлагает больше возможностей для настройки. По аналогии с предыдущими главами, в которых рассматривались виджеты, я начну с того, как создать этот виджет, а затем расскажу о свойствах, методах и событиях, которые он поддерживает. Глава заканчивается примером, демонстрирующим применение виджета Tabs для представления форм в виде отдельных вкладок, что особенно эффективно в случае крупных форм, требующих ввода многочисленных данных. Перечень тем, рассматриваемых в данной главе, приведен в табл. 20.1.

Таблица 20.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Создание виджета Tabs	Определите структуру для ярлыков и вкладок и вызовите метод `tabs()`	1
Получение содержимого вкладки с помощью Ajax	Укажите в атрибуте `href` элемента `a` вкладки ссылку на HTML-документ, который должен отображаться на панели содержимого	2, 3
Использование содержимого вкладки в формате JSON	Используйте опцию `ajaxOptions` для применения опций `dataType` и `dataFilter` при работе с Ajax	4, 5
Обработка ошибок в Ajax-запросах содержимого вкладок	Определите функцию-обработчик для события Ajax `error` с помощью опции `ajaxOPptions`	6
Отображение сообщений для пользователя в процессе загрузки содержимого посредством Ajax-запросов	Добавьте в элемент `a` вкладки элемент `span` и используйте опцию `spinner`	7, 8
Отключение отдельных вкладок	Используйте опцию `disabled`	9
Изменение события, активизирующего вкладку	Используйте опцию `event`	10
Разрешение одновременного отключения всех вкладок	Используйте опцию `collapsible`	11
Добавление и удаление вкладок программным путем	Используйте методы `add()` и `remove()`	12–14
Изменение элемента, используемого для создания панелей содержимого	Используйте опцию `panelTemplate`	15

Окончание табл. 20.1

Задача	Решение	Листинг
Принудительная загрузка дистанционного содержимого	Используйте метод `load()`	16
Изменение панели или источника содержимого для вкладки	Используйте метод `url()`	17
Автоматический циклический обход вкладок	Используйте метод `rotate()`	18, 19
Отображение формы в нескольких вкладках	Разбейте форму на несколько разделов с помощью элементов `div`, добавьте структуру ярлыка и вызовите метод `tabs()`	20–22
Проверка заполнения формы, отображаемой в нескольких вкладках	Используйте события `show` и `select`	23

Создание виджета Tabs

Для создания виджета Tabs используется метод `tabs()`. Как и в случае виджета Accordion, для применения этого метода потребуется отдельная структура HTML-элементов. Пример такой структуры приведен в листинге 20.1.

Листинг 20.1. Создание виджета jQuery UI Tabs

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"},
                        {"name":"Примулы","product":"primula"},
                        {"name":"Подснежники",
                            "product":"snowdrop"},
                        {"name":"Гвоздики","product":"carnation"},
                        {"name":"Лилии","product":"lily"},
                        {"name":"Орхидеи","product":"orchid"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 3).appendTo("#tab1");
            elems.slice(3, 6).appendTo("#tab2");
            elems.slice(6).appendTo("#tab3");
```

```
            $('#tabs').tabs();

            $('button').button();
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="tabs">
            <ul>
                <li><a href="#tab1">Ряд 1</a>
                <li><a href="#tab2">Ряд 2</a>
                <li><a href="#tab3">Ряд 3</a>
            </ul>
            <div id="tab1"></div>
            <div id="tab2"></div>
            <div id="tab3"></div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button></div>
    </form>
</body>
</html>
```

Элемент, к которому вы собираетесь применить метод `tabs()`, должен содержать элементы двух типов. Элементы первого типа — это *элементы содержимого*, т.е. элементы, содержимое которых должно отображаться в отдельных вкладках. Ко второму типу относятся *структурные элементы*, содержащие информацию, которая используется для создания структуры вкладок.

В качестве контейнеров содержимого используются элементы `div`. В данном примере используются три элемента `div`, каждый из которых будет содержать один ряд элементов с информацией о цветочной продукции, аналогично тому, как это делалось во всех предыдущих примерах.

```
<div id="tab1"></div>
<div id="tab2"></div>
<div id="tab3"></div>
```

Очень важно, чтобы каждый элемент содержимого был снабжен идентификатором (атрибутом `id`) и виджет мог находить элемент, который требуется отобразить. Для создания необходимой структуры используются элементы `li`, каждый из которых должен содержать элемент `a`.

```
<ul>
    <li><a href="#tab1">Ряд 1</a>
    <li><a href="#tab2">Ряд 2</a>
    <li><a href="#tab3">Ряд 3</a>
</ul>
```

Количество элементов `li` определяет количество вкладок. Содержимое элемента `a` используется в качестве ярлыка вкладки, а атрибут `href` указывает, к какому именно элементу содержимого (вкладке) относится данный ярлык.

Совет. Для динамической генерации содержимого вкладок я использовал подключаемый модуль шаблонов данных, поскольку это дает возможность более отчетливо показать структуру. Содержимое может быть статическим или, о чем будет говориться в следующем разделе, его можно получать с сервера в динамическом режиме.

Каким образом использованная в примере структура HTML-элементов отображается в окне браузера, показано на рис. 20.1.

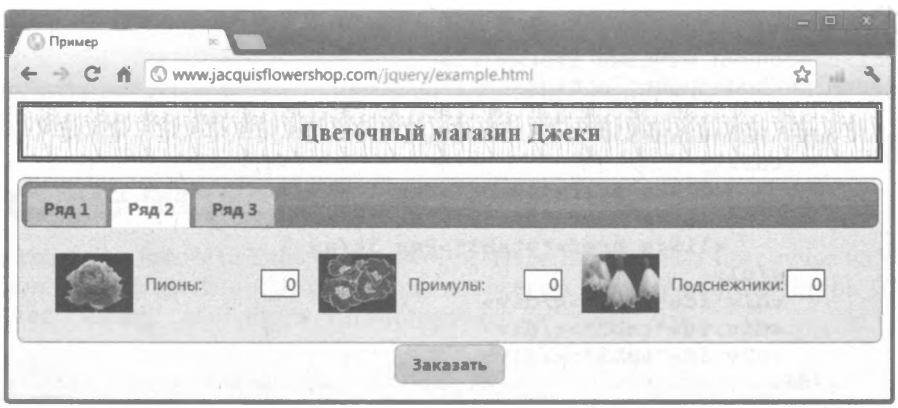


Рис. 20.1. Создание вкладок jQuery UI

Вкладки являются хорошо известной метафорой интерфейса пользователя. Щелчок на вкладке приводит к отображению соответствующего элемента содержимого. Как и виджет Accordion с его структурой панелей, напоминающей клавиатуру аккордеона и раздвигающейся подобно мехам гармошки, виджет Tabs обеспечивает возможность относительно компактного представления больших объемов содержимого, что позволяет сконцентрировать внимание на наиболее важной информации. Это означает, что необходимо тщательно продумывать, каким образом вкладки и их содержимое должны соотноситься между собой. Ваша задача состоит в том, чтобы путем группирования информации избавить пользователя от необходимости часто переходить с одной вкладки на другую для получения доступа к нужным данным. Как и в случае любого пользовательского интерфейса, это требует глубокого понимания задач, решаемых пользователем, и того, как организован весь его рабочий процесс (а не только ваша система).

Получение содержимого вкладок с помощью Ajax

Одной из замечательных особенностей вкладок jQuery UI является возможность получения содержимого с помощью Ajax. Для этого достаточно указать URL-адрес источника в атрибуте `href` соответствующего элемента. Чтобы продемонстрировать, как это работает, я создал документ под названием `tabflowers.html`, содер-

жимое которого представлено в листинге 20.2. Вкладка, которая получает свое содержимое посредством Ajax-запросов, называется *дистанционной вкладкой*.

Листинг 20.2. Содержимое файла `tabflowers.html`

```
<div>
    <div class="dcell">
        <img src="astor.png"/>
        <label for="astor">Астры:</label>
        <input name="astor" value="0" />
    </div>
    <div class="dcell">
        <img src="daffodil.png"/>
        <label for="daffodil">Нарциссы:</label>
        <input name="daffodil" value="0" />
    </div>
    <div class="dcell">
        <img src="rose.png"/>
        <label for="rose">Розы:</label>
        <input name="rose" value="0" />
    </div>
</div>
<div>
    <div class="dcell">
        <img src="peony.png"/>
        <label for="peony">Пионы:</label>
        <input name="peony" value="0" />
    </div>
    <div class="dcell">
        <img src="primula.png"/>
        <label for="primula">Примулы:</label>
        <input name="primula" value="0" />
    </div>
    <div class="dcell">
        <img src="snowdrop.png"/>
        <label for="snowdrop">Подснежники:</label>
        <input name="snowdrop" value="0" />
    </div>
</div>
```

Чтобы не усложнять пример, я использую те же структуру и содержимое, что и в генерируемых элементах содержимого. Использование файла `tabflowers.html` в качестве содержимого вкладок показано в листинге 20.3.

Листинг 20.3. Получение содержимого вкладки посредством Ajax

```
...
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="tabs">
            <ul>
                <li><a href="#tabflowers.html">Содержимое Ajax</a>
                <li><a href="#tab1">Ряд 1</a>
                <li><a href="#tab2">Ряд 2</a>
                <li><a href="#tab3">Ряд 3</a>
```

```
            </ul>
            <div id="tab1"></div>
            <div id="tab2"></div>
            <div id="tab3"></div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button></div>
    </form>
</body>
```

Вместо того чтобы использовать настройки метода `tabs()`, вы изменяете структуру элемента. В данном примере я добавил новую вкладку, назвав ее Содержимое Ajax, и указал URL-адрес источника содержимого, которое должно быть загружено. Результат представлен на рис. 20.2.

Совет. В создании элемента содержимого для дистанционной вкладки нет необходимости. Виджет Tabs автоматически сделает это вместо вас.

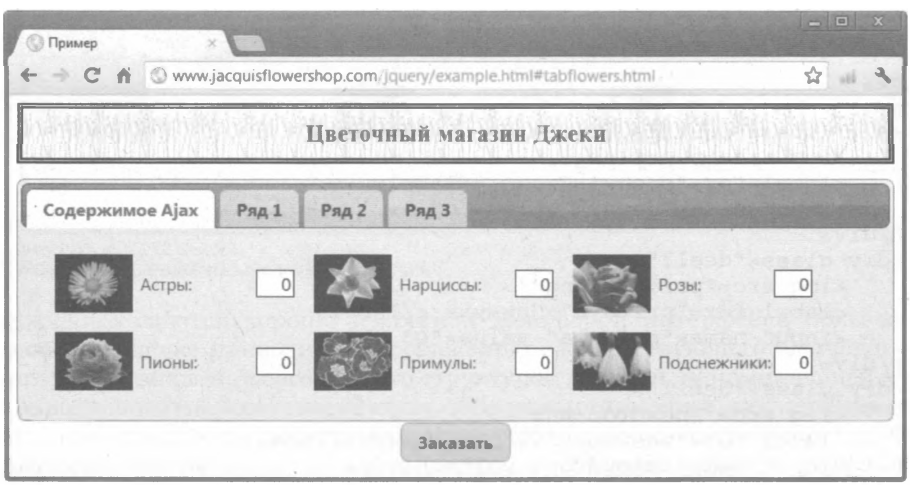


Рис. 20.2. Получение содержимого вкладки посредством Ajax

Совет. По умолчанию загружаться во вкладки с помощью Ajax могут только HTML-документы, однако, как будет показано в одном из следующих разделов, для этого пригодны также данные в формате JSON.

Настройка виджета Tabs

На первый взгляд, вкладки виджета Tabs могут показаться простой разновидностью панелей виджета Accordion (см. главу 19), характеризующихся вертикальным расположением. Действительно, у этих виджетов есть общие характеристики, но для вкладок Tabs (далее просто *вкладки*) предусмотрено намного больше настраиваемых свойств и конфигурационных параметров. Свойства, поддерживаемые виджетом Tabs, приведены в табл. 20.2. Примеры их использования для настройки виджета описываются в следующих разделах.

Таблица 20.2. Свойства виджета Tabs

Свойство	*Описание*
`ajaxOptions`	Позволяет устанавливать и получать значения конфигурационных параметров для Ajax-запросов
`cache`	Если эта опция равна `true`, то полученное в результате Ajax-запроса содержимое будет кешироваться, так что при следующем открытии вкладки запрос не будет повторно выполняться. Значение по умолчанию — `false`, которое означает, что содержимое дистанционной вкладки будет загружаться с сервера при каждом ее открытии
`collapsible`	Если эта опция равна `true`, то пользователь не будет иметь возможности оставить невыбранными все вкладки. Значение по умолчанию — `false`, которое означает, что одна из вкладок всегда должна быть активна (открыта)
`disabled`	Установка значения `true` или `false` означает соответственно отключение или включение функциональности вкладок. Если в качестве значения задан массив чисел, то они указывают индексы отключаемых вкладок
`event`	Позволяет получить или задать событие, которое делает вкладку активной. По умолчанию таким событием является `click`, т.е. вкладка активизируется после выполнения на ней щелчка
`fx`	Определяет эффекты, которые должны использоваться при анимации процессов открытия и закрытия вкладок. Значение по умолчанию — `null`, означающее, что эффекты не используются. Эффекты jQuery UI описаны в главе 34
`panelTemplate`	Определяет шаблон, в соответствии с которым должны генерироваться элементы содержимого, создаваемые программным путем. По умолчанию для этого используется элемент `div`. Более подробно о добавлении вкладок программным путем речь идет в одном из следующих разделов
`selected`	Позволяет получить или задать индекс активной вкладки
`spinner`	Позволяет получить или задать текст, отображаемый для пользователя во время загрузки содержимого дистанционных вкладок. Более подробно об использовании этой опции будет говориться в одном из следующих разделов
`tabTemplate`	Определяет шаблон, в соответствии с которым должны генерироваться структурные элементы, создаваемые программным путем. Более подробно о добавлении вкладок программным путем речь идет в одном из следующих разделов

Настройка Ajax-запросов

С помощью опции `ajaxOptions` можно предоставить объект отображения, содержащий конфигурационные параметры, которые должны использоваться при выполнении Ajax-запросов для дистанционных вкладок. С помощью этой опции можно установить любой из параметров, описанных в главах 14 и 15. Чаще всего я использую опцию `ajaxOptions` для указания того, что содержимое дистанционных вкладок должно генерироваться в формате JSON, а не в виде HTML-кода. Для демонстрации этой методики будет использоваться файл `mydata.json`, содержимое которого представлено в листинге 20.4. Это тот же файл, который мы использовали в главе 14.

Определившись с источником данных, обратимся к рассмотрению опции `ajaxOptions`, пример использования которой приведен в листинге 20.5.

Листинг 20.4. Содержимое файла `mydata.json`

```
[{"name":"Астры","product":"astor","stocklevel":"10",
  "price":"2.99"},
 {"name":"Нарциссы","product":"daffodil","stocklevel":"12",
  "price":"1.99"},
 {"name":"Розы","product":"rose","stocklevel":"2",
  "price":"4.99"},
{"name":"Пионы","product":"peony","stocklevel":"0",
  "price":"1.50"},
 {"name":"Примулы","product":"primula","stocklevel":"1",
  "price":"3.12"},
 {"name":"Подснежники","product":"snowdrop","stocklevel":"15",
  "price":"0.99"}]
```

Листинг 20.5. Использование опции `ajaxOptions` для работы с данными в формате JSON

```
...
$('#tabs').tabs({
    ajaxOptions: {
        dataType: "html",
        dataFilter: function(result){
            var data = $.parseJSON(result).slice(0, 3);
            return $('<div></div>')
                .append($('#flowerTmpl').tmpl(data)).html();
        }
    }
});
...
```

Чтобы получить правильный результат, необходимо использовать две опции, конфигурирующие запросы Ajax. Одна из них — это опция `dataFilter`, определяющая функцию, которая преобразует данные в формате JSON в объект JavaScript (с помощью метода `parseJSON()`, описанного в главе 33), применяет шаблон данных, осуществляет разбивку массива HTML-элементов на группы и возвращает содержимое в виде HTML-строки. Все перечисленные средства являются стандартными, и их использование демонстрировалось в главах 14–16.

Вторая из упомянутых опций — это опция `dataType`. Поскольку сервер информирует браузер о том, что он отправляет данные в формате JSON (это делается с помощью HTTP-заголовка `Content-Type`), средства Ajax, поддерживаемые в jQuery, сталкиваются с проблемой, пытаясь обработать HTML-строку, которую генерирует функция, заданная в опции `dataFilter`. Для решения этой проблемы привлекается опция `dataType`, с помощью которой мы сообщаем jQuery, что данные следует обрабатывать как HTML-разметку даже в том случае, если сервер декларирует их как данные в формате JSON.

Разумеется, для того чтобы можно было воспользоваться этими изменениями, необходимо указать в качестве источника данных для дистанционной вкладки файл JSON.

```
...
<ul>
    <li><a href="mydata.json">Содержимое Ajax</a>
    <li><a href="#tab1">Ряд 1</a>
    <li><a href="#tab2">Ряд 2</a>
```

```
    <li><a href="#tab3">Ряд 3</a>
</ul>
...
```

Таким образом, после внесения описанных изменений виджет Tabs будет запрашивать файл JSON, тогда как опции, заданные опцией `ajaxOptions`, обеспечат преобразование данных JSON в HTML с использованием шаблона данных. Результат будет отображаться на панели дистанционной вкладки, как показано на рис. 20.3.

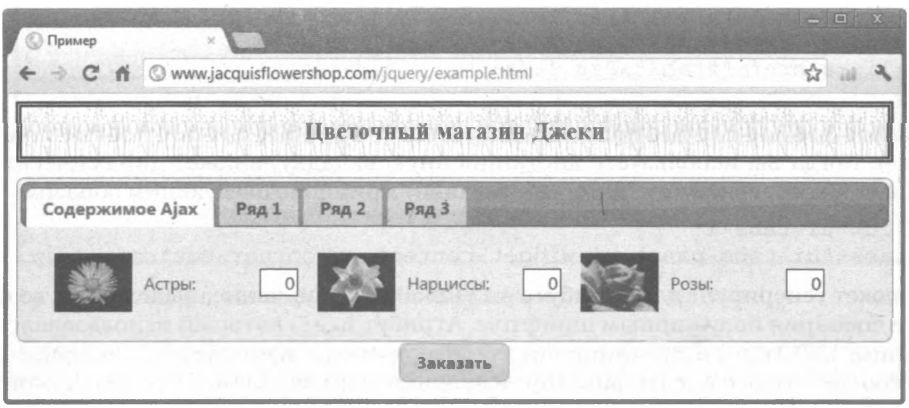


Рис. 20.3. Использование JSON для генерации содержимого дистанционных вкладок

Обработка ошибок Ajax

По умолчанию виджет Tabs справляется с проблемами Ajax, корректно игнорируя их, что является далеко не идеальным решением. К счастью, вы можете применить свое "ноу-хау" в виде поддерживаемых в jQuery средств Ajax в сочетании с опцией `ajaxOptions` для того, чтобы должным образом проинформировать пользователя, если что-то пойдет не так. Соответствующий демонстрационный пример приведен в листинге 20.6.

Листинг 20.6. Информирование пользователя об ошибках, возникающих в процессе загрузки содержимого дистанционной вкладки средствами Ajax

```
...
$('#tabs').tabs({
    ajaxOptions: {
        dataType: "html",
        dataFilter: function(result){
            var data = $.parseJSON(result).slice(0, 3);
            return $('<div></div>')
                .append($('#flowerTmpl').tmpl(data)).html();
        },
        error: function(jqxr, status, index, anchor) {
            $(anchor.hash).text("Извините. Во время загрузки
                содержимого произошла ошибка");
        }
    }
});
...
```

Кроме тех аргументов, которые обычно используются средствами jQuery Ajax, виджет Tabs передает в функцию обработки ошибок дополнительный аргумент. Им является элемент, используемый в качестве заголовка дистанционной вкладки, в которой встретилась ошибка. Для создания ситуации, ведущей к возникновению ошибки, я изменил имя файла, который указывается в качестве источника дистанционного содержимого.

```
<ul>
    <li><a href="mydata.jsonX">Содержимое Ajax</a>
    <li><a href="#tab1">Ряд 1</a>
    <li><a href="#tab2">Ряд 2</a>
    <li><a href="#tab3">Ряд 3</a>
</ul>
```

Такого файла не существует, поэтому виджет Tabs безусловно столкнется с проблемой. Когда вы используете дистанционную вкладку, виджет динамически создает элемент содержимого, который выглядит примерно следующим образом:

```
<div id="ui-tabs-1"
   class="ui-tabs-panel ui-widget-content ui-corner-bottom"></div>
```

Виджет генерирует для атрибута `id` уникальное значение, выделенное во фрагменте сценария полужирным шрифтом. Атрибут `href`, который использовался для указания URL-адреса дистанционного содержимого, изменяется для приведения его в соответствие с идентификатором вновь созданного элемента содержимого.

```
<li class="ui-state-default ui-corner-top ui-tabs-selected
    ui-state-active">
    <a href="#ui-tabs-1">Содержимое Ajax</a>
</li>
```

Здесь элемент `a` — этот тот элемент, который передается функции обработки ошибок посредством опции `ajaxOptions`. Это означает, что можно использовать свойство `hash`, которое определено объектом DOM, представляющим элемент `a`, для получения идентификатора элемента содержимого, чтобы затем отобразить сообщение для пользователя с помощью метода jQuery `text()`.

```
$(anchor.hash).text("Извините. Во время загрузки
                     содержимого произошла ошибка");
```

Результат представлен на рис. 20.4. Отображать сообщение об ошибке именно в элементе содержимого вовсе не обязательно, но, с моей точки зрения, для большинства веб-приложений такой подход является наиболее естественным, главным образом потому, что именно здесь пользователь ожидает появления содержимого, о котором информирует заголовок вкладки.

Вывод сообщений Ajax с помощью опции `spinner`

Во время загрузки содержимого дистанционной вкладки виджет Tabs может отображать вместо ее заголовка сообщение для пользователя. Чтобы активизировать это средство, содержимое элемента `a` необходимо "обернуть" элементом `span`, как показано в листинге 20.7.

Листинг 20.7. Включение средства Ajax Spinner

```
...
<ul>
    <li><a href="tabflowers.html"><span>Содержимое Ajax</span></a>
```

```
    <li><a href="#tab1">Ряд 1</a>
    <li><a href="#tab2">Ряд 2</a>
    <li><a href="#tab3">Ряд 3</a>
</ul>
...
```

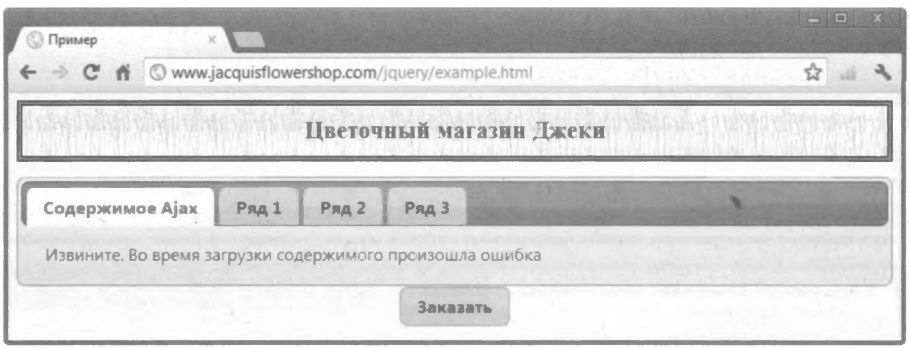


Рис. 20.4. Отображение сообщения об ошибке для пользователей

По умолчанию в течение всего времени, пока выполняется запрос, в качестве упомянутого сообщения используется строка `Loading...` (точнее, строка сообщения имеет вид `<em>Loading&#8230;</em>`). Результат представлен на рис. 20.5.

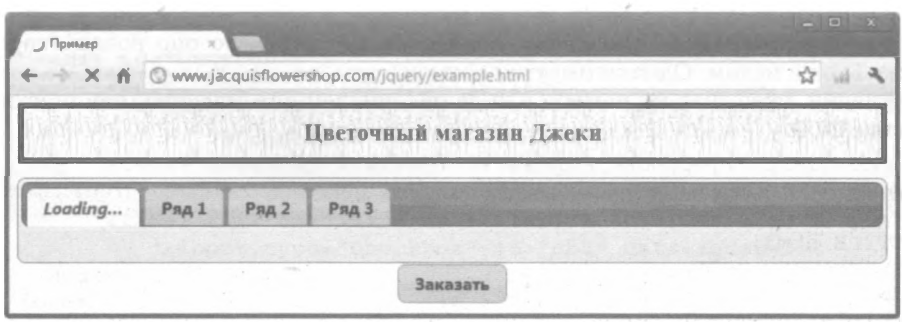


Рис. 20.5. Виджет Tabs, отображающий сообщение о загрузке данных

Совет. Очень важно, чтобы текст сообщения находился внутри элемента `span`. Многие допускают ошибку, вставляя пустой элемент `span` перед ярлыком вкладки или после него. В этом случае текст сообщения, предусмотренного по умолчанию, все так же будет добавляться в элемент `span` с началом загрузки, но по ее окончании он не будет удален.

Текст сообщения, отображаемого в процессе загрузки данных, можно изменить с помощью опции `spinner`, как показано в листинге 20.8.

Листинг 20.8. Использование опции `spinner` для отображения текста сообщения во время загрузки данных

```
...
$('#tabs').tabs({
    spinner: "<em>Загружаются данные...</em>"
```

```
});
...
```

Не указывайте в опции spinner слишком длинные сообщения. Чтобы отобразить сообщение, размер ярлыка вкладки должен будет соответственным образом измениться, что может приводить к скачкообразному увеличению размера ярлыка в начале загрузки данных и столь же резкому его уменьшению после того, как запрос будет выполнен (рис. 20.6).

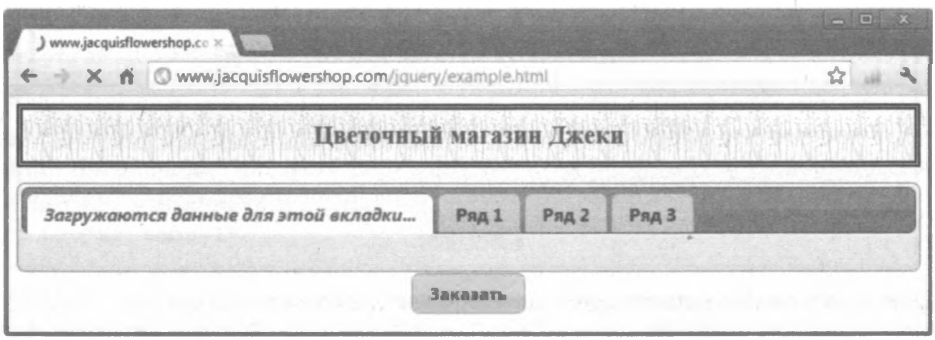


Рис. 20.6. Изменение размера ярлыка вкладки в соответствии с длиной сообщения

Отключение отдельных вкладок

Если опции disable присвоено логическое значение, то оно воздействует на виджет Tabs в целом. Однако имеется возможность включать и отключать отдельные вкладки, указывая их номера в виде целочисленного массива, как показано в листинге 20.9.

Листинг 20.9. Включение и отключение отдельных вкладок

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css"> #buttonDiv {margin: 5px}</style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#tabs').tabs({
                select: function(event, ui) {
                    $('input:checkbox').button("enable")
                        .filter('#cb' + ui.index)
                        .button("disable")
                }
            });
```

```
            $('input:checkbox').button().click(function() {
                var disabledPositions = [];
                $('input:checkbox[checked]')
                    .each(function(index, elem) {
                        disabledPositions.push($(this)
                        .data("index"));
                })
                $('#tabs')
                    .tabs("option", "disabled", disabledPositions)
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="tabs">
            <ul>
                <li><a href="#tab1">Вкладка 1</a>
                <li><a href="#tab2">Вкладка 2</a>
                <li><a href="#tab3">Вкладка 3</a>
            </ul>
            <div id="tab1">Это содержимое для вкладки 1</div>
            <div id="tab2">Это содержимое для вкладки 2</div>
            <div id="tab3">Это содержимое для вкладки 3</div>
        </div>
        <div id="buttonDiv">
            <label for="cb0">Вкладка 1</label>
                <input type="checkbox" id="cb0" data-index=0
                    disabled>
            <label for="cb1">Вкладка 2</label>
                <input type="checkbox" id="cb1" data-index=1>
            <label for="cb2">Вкладка 3</label>
                <input type="checkbox" id="cb2" data-index=2>
        </div>
    </form>
</body>
</html>
```

В этом документе мы создаем виджет Tabs со статическим содержимым и добавляем набор флажков, преобразуя их в кнопки-переключатели jQuery UI. Щелчок на кнопке приводит к включению или отключению соответствующей вкладки. Кроме того, мы используем событие `select` для отключения кнопки, когда соответствующая ей вкладка становится активной. (О событиях вкладок далее будет говориться более подробно.) Результат представлен на рис. 20.7.

Изменение типа события, активизирующего вкладку

По умолчанию виджет Tabs реагирует на событие `click`. Это означает, что для активизации вкладки пользователь должен выполнить щелчок на ней. Опция `event` позволяет указать другое событие, наступление которого будет приводить к активизации вкладки. Чаще всего в этой роли выступают другие события мыши, как показано в листинге 20.10.

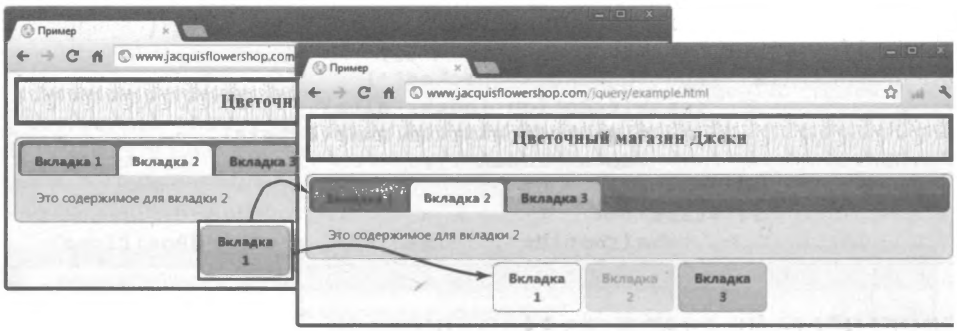


Рис. 20.7. Отключение и включение вкладок с помощью щелчков на кнопках

Листинг 20.10. Изменение типа события, активизирующего вкладку

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css"> #buttonDiv {margin: 5px}</style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#tabs').tabs({
                event: "mouseover"
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="tabs">
            <ul>
                <li><a href="#tab1">Вкладка 1</a>
                <li><a href="#tab2">Вкладка 2</a>
                <li><a href="#tab3">Вкладка 3</a>
            </ul>
            <div id="tab1">Это содержимое для вкладки 1</div>
            <div id="tab2">Это содержимое для вкладки 2</div>
            <div id="tab3">Это содержимое для вкладки 3</div>
        </div>
    </form>
</body>
</html>
```

В этом примере указано событие `mouseover`, т.е. переключение вкладок в этом виджете будет осуществляться при наведении указателя мыши на ярлык соответствующей вкладки.

Совет. Я уже давал совет относительно умеренного использования описанного подхода в случае виджета Accordion, и сейчас также рекомендую поступать аналогичным образом. Внешне этот эффект кажется привлекательным, но он может раздражать пользователей, поскольку заставляет их постоянно следить за тем, чтобы указатель мыши случайно не оказался над ярлыком другой вкладки, что приведет к смене отображаемого содержимого.

Использование свертываемых вкладок

Используя опцию `collapsible`, можно создать своего рода гибрид вкладок виджета Tabs и панелей виджета Accordion, как показано в листинге 20.11.

Листинг 20.11. Использование опции `collapsible`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#tabs').tabs({
            collapsible: true
        });
    });
</script>
...
```

Если опция `collapsible` равна `true`, то щелчок на активной вкладке приводит к ее свертыванию, точно так же, как в случае панели виджета Accordion. Такой переход показан на рис. 20.8.

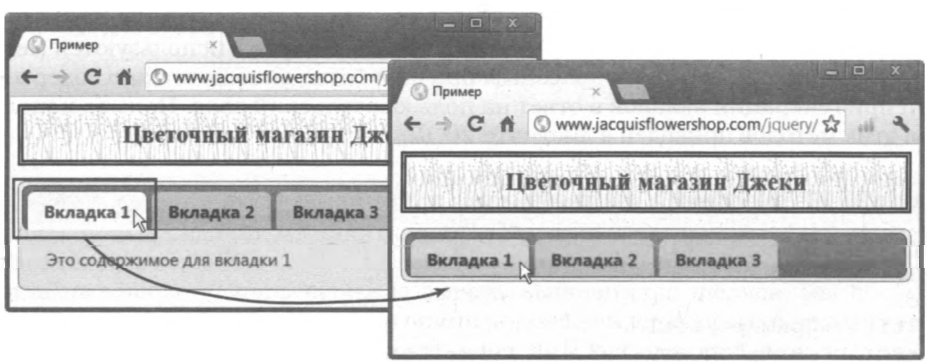


Рис. 20.8. Свертывание активной вкладки

Совет. Я привел пример использования этой опции лишь для полноты обсуждения, но сам никогда не использую ее в своих проектах, поскольку считаю, что этот эффект не относится к числу интуитивно понятных для пользователей и лишь создает помехи в работе.

Использование методов виджета Tabs

Методы, поддерживаемые виджетом jQuery UI Tabs, перечислены в табл. 20.3. Использование наиболее полезных из них будет продемонстрировано в следующих разделах.

Таблица 20.3. Методы виджета Tabs

Метод	*Описание*
`tabs("destroy")`	Полностью удаляет функциональность виджета Tabs из базового HTML-элемента
`tabs("disable")`	Приостанавливает работу всего виджета или отдельных вкладок. (См. пример использования соответствующей опции в одном из предыдущих разделов.)
`tabs("enable")`	Возобновляет работу ранее приостановленного виджета или отдельных вкладок
`tabs("option")`	Позволяет изменить одну или несколько опций. Более подробное описание настройки конфигурационных параметров виджетов jQuery UI на примере виджета Button приведено в главе 18
`tabs("add")`	Добавляет новую вкладку
`tabs("remove")`	Удаляет вкладку
`tabs("select")`	Активизирует вкладку
`tabs("load")`	Осуществляет принудительную загрузку содержимого вкладки
`tabs("url")`	Изменяет URL-адрес источника содержимого дистанционной вкладки
`tabs("length")`	Возвращает количество вкладок в виджете
`tabs("abort")`	Отменяет все активные Ajax-запросы для дистанционных вкладок
`tabs("rotate")`	Указывает виджету Tabs на необходимость циклического обхода вкладок

Добавление и удаление вкладок

Для добавления и удаления вкладок программным путем используются методы `add` и `remove`. Это может быть полезным при обработке динамического содержимого или при генерации вкладок в ответ на пользовательский ввод. Пример использования этих методов приведен в листинге 20.12.

Листинг 20.12. Добавление и удаление вкладок программным путем

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css"> #buttonDiv {margin: 5px}</style>
    <style type="text/css">
        input {width: 150px; text-align: left}
        #dc1 {margin: 5px}
```

```
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#tabs').tabs();

            $('button').button().click(function(e) {
                var tabsElem = $('#tabs');
                if (this.id == "add") {
                    tabsElem.tabs("add", "tabflowers.html",
                        $('#tabLabel').val());
                } else {
                    tabsElem.tabs("remove", tabsElem
                        .tabs("option", "selected"))
                }
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div id="dc1" class="ui-widget">
        <label for="tabLabel">Имя вкладки: </label>
            <input id="tabLabel"/>
        <button id="add">Добавить вкладку</button>
        <button id="remove">Удалить активную вкладку</button>
    </div>

    <div id="tabs">
        <ul>
            <li><a href="#tab1">Вкладка 1</a>
            <li><a href="#tab2">Вкладка 2</a>
            <li><a href="#tab3">Вкладка 3</a>
        </ul>
        <div id="tab1">Это содержимое для вкладки 1</div>
        <div id="tab2">Это содержимое для вкладки 2</div>
        <div id="tab3">Это содержимое для вкладки 3</div>
    </div>
</body>
</html>
```

В этом примере добавлены элемент `input` и два элемента `button`, позволяющие добавлять новые вкладки и удалять существующие. Эти дополнения документа отображены на рис. 20.9.

Когда на кнопке Удалить активную вкладку выполняется щелчок, мы получаем индекс активной вкладки с помощью опции `selected` и передаем его в качестве аргумента методу `remove`.

```
tabsElem.tabs("remove", tabsElem
    .tabs("option", "selected"))
```

Виджет Tabs удаляет вкладку с указанным индексом и активизирует следующую вкладку. Если дистанционная вкладка была последней вкладкой виджета, то активизируется текущая последняя вкладка.

Щелчок на кнопке Добавить вкладку приводит к вызову метода `add`.

```
tabsElem.tabs("add", "tabflowers.html",
    $('#tabLabel').val());
```

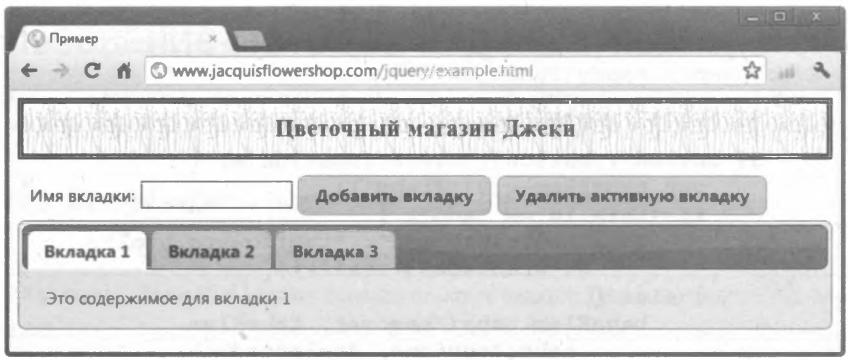


Рис. 20.9. Включение в документ новых элементов, обеспечивающих добавление и удаление вкладок

После аргумента `add` указаны URL-адрес источника содержимого для вкладки и ярлык, который мы хотим использовать. В данном случае для получения содержимого элемента `input` используется метод `val()`. В этом примере URL-адрес ссылается на документ `tabflowers.html`. Конечным результатом является создание дистанционной вкладки.

Совет. По умолчанию новая вкладка добавляется в виджет в качестве последней по счету. Такой порядок следования вкладок можно изменить, предоставив методу `add` дополнительный аргумент, указывающий на отсчитываемый от нуля номер позиции, в которую должна быть вставлена новая вкладка.

На рис. 20.10 показано, что произойдет, если ввести в текстовом поле текст `Новая вкладка` и щелкнуть на кнопке **Добавить вкладку**

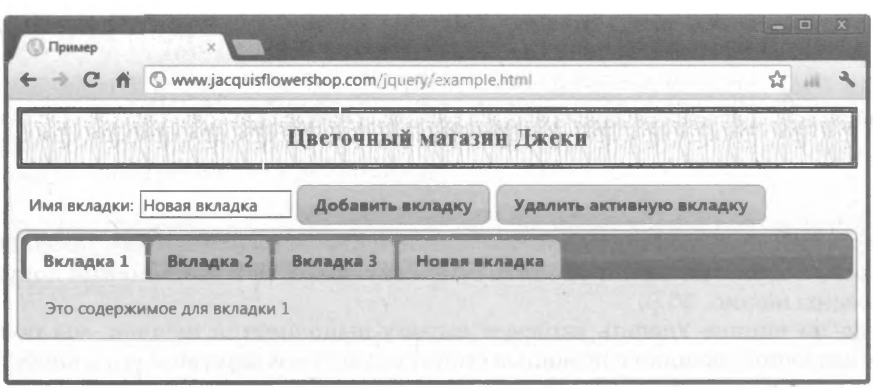


Рис. 20.10. Добавление вкладки программным путем

По умолчанию новая вкладка сразу не делается активной. Самый простой способ активизировать ее заключается в обработке события `add`, как показано в листинге 20.13. О том, какие события поддерживаются виджетом Tabs, речь пойдет далее, но вы уже сейчас должны понимать, что событие `add` играет весьма полезную роль в контексте метода `add`.

Листинг 20.13. Автоматическая активизация вновь созданной вкладки

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#tabs').tabs({
            add: function(event, ui) {
                $(this).tabs("select", ui.index);
            }
        });

        $('button').button().click(function(e) {
            var tabsElem = $('#tabs');
            if (this.id == "add") {
                tabsElem.tabs("add", "tabflowers.html",
                    $('#tabLabel').val());
            } else {
                tabsElem.tabs("remove", tabsElem
                    .tabs("option", "selected"))
            }
        })
    });
</script>
...
```

Для предоставления информации о вновь созданной вкладке виджет Tabs использует объект `ui`. Свойство `index` этого объекта возвращает индекс новой вкладки, использование которого в сочетании с методом `select` обеспечивает активизацию вкладки сразу же после ее создания.

Добавление вкладок со статическим содержимым

Событие `add` и объект `ui` также могут использоваться для добавления вкладок, содержимое которых является статическим, как показано в листинге 20.14.

Листинг 20.14. Добавление вкладок со статическим содержимым

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#tabs').tabs({
            add: function(event, ui) {
                $(this).tabs("select", ui.index);
                $(ui.panel).html("Это <b>новая</b> панель")
            }
        });

        var newTabCount = 0;

        $('button').button().click(function(e) {
            var tabsElem = $('#tabs');
            if (this.id == "add") {
                tabsElem.tabs("add", "#" + (newTabCount++),
                    $('#tabLabel').val());
            } else {
                tabsElem.tabs("remove", tabsElem
```

```
                    .tabs("option", "selected"))
            }
        })
    });
</script>
...
```

Чтобы создать статическую вкладку, необходимо указать идентификатор фрагмента в качестве URL-адреса точно так же, как это делается при определении структуры вкладок с использованием HTML-элементов. Важно лишь убедиться в том, что данный фрагмент отличается от вкладок. В случае дублирования идентификаторов виджет jQuery UI Tabs столкнется с проблемами при активизации вкладок программным путем.

Статические вкладки создаются без содержимого. К счастью, объект `ui`, который передается функции-обработчику события `add`, определяет свойство `panel`, которое возвращает HTML-элемент, создаваемый виджетом Tabs во время вызова метода `add`. Для добавления содержимого в эту панель подойдет любая из методик jQuery, пригодных для этих целей. В примере для вставки простого сообщения используется метод `html()`.

```
$(ui.panel).html("Это <b>новая</b> панель")
```

Конечный результат представлен на рис. 20.11.

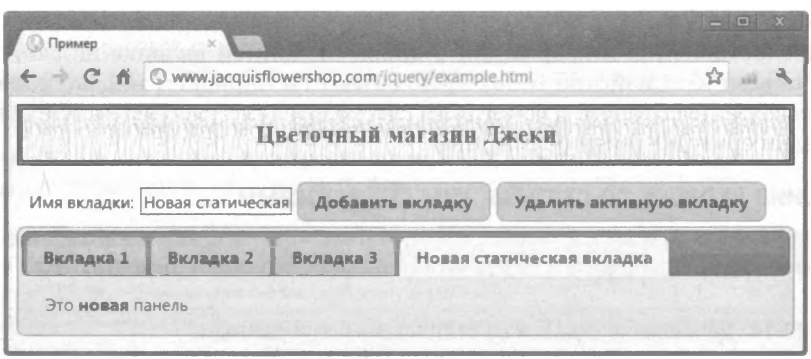


Рис. 20.11. Добавление содержимого во вновь созданную статическую вкладку

Другой возможный подход состоит в использовании опции `panelTemplate` для изменения шаблона, следуя которому, виджет создает вкладки. Соответствующий пример приведен в листинге 20.15.

Листинг 20.15. Изменение шаблона вкладки

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#tabs').tabs({
            add: function(event, ui) {
                $(this).tabs("select", ui.index);
            },
            panelTemplate: "<div>Это содержимое, используемое
                <b>по умолчанию</b></div>"
```

```
        });
         var newTabCount = 0;
         $('button').button().click(function(e) {
             var tabsElem = $('#tabs');
             if (this.id == "add") {
                 tabsElem.tabs("add", "#" + (newTabCount++),
                     $('#tabLabel').val());
             } else {
                 tabsElem.tabs("remove", tabsElem
                     .tabs("option", "selected"))
             }
         })
    });
</script>
...
```

По умолчанию в качестве шаблона используется пустой элемент `div`. В этом примере он заменен элементом `div`, содержащим простое сообщение. Конечный результат приведен на рис. 20.12. Разумеется, вы по-прежнему можете использовать событие `add` для указания дополнительного содержимого или замены содержимого, используемого по умолчанию, но замена шаблона предоставляет превосходную отправную точку для создания вкладок.

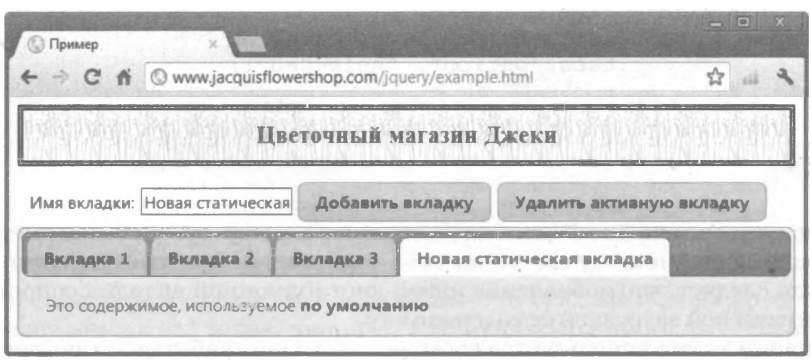


Рис. 20.12. Определение содержимого, используемого по умолчанию, для программно добавляемых статических вкладок

Совет. Вы также можете изменить HTML-разметку, используемую для создания ярлыков новых вкладок. По умолчанию используется следующий шаблон:

```
<li><a href="#{href}"><span>#{label}</span></a></li>
```

Однако его можно изменить с помощью опции `tabTemplate`. Обратите внимание на использование заполнителей `href` и `label`. Измените их, если хотите создать собственные ярлыки.

Управление Ajax-запросами дистанционной вкладки

Для управления способом загрузки содержимого дистанционных вкладок можно использовать ряд методов. Одним из них, чаще всего используемым мною, является метод `load()`, вызов которого приводит к немедленной генерации Ajax-запроса для получения содержимого вкладки. Это очень выручает меня при динамическом

создании вкладок, когда я не хочу, чтобы пользователь дожидался окончания загрузки содержимого и лишь затем активизировал вкладку. Пример использования метода `load()` представлен в листинге 20.16.

Листинг 20.16. Принудительная загрузка содержимого дистанционной вкладки

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#tabs').tabs({
            add: function(event, ui) {
                $(this).tabs("load", ui.index);
            },
            load: function(event, ui) {
                $(this).tabs("select", ui.index);
            }
        });

        $('button').button().click(function(e) {
            var tabsElem = $('#tabs');
            if (this.id == "add") {
                tabsElem.tabs("add", "tabflowers.html",
                    $('#tabLabel').val());
            } else {
                tabsElem.tabs("remove", tabsElem
                    .tabs("option", "selected"))
            }
        })
    });
</script>
...
```

Аргументом метода `load` здесь служит индекс интересующей нас дистанционной вкладки. В данном случае метод `load` вызывается при наступлении события `add`, откуда следует, что добавление новой дистанционной вкладки сопровождается автоматической загрузкой ее содержимого.

В сценарии также используется событие `load`. Оно срабатывает по завершении загрузки содержимого дистанционной вкладки. Мы используем это событие для активизации вкладки после того, как выполнится Ajax-запрос, порожденный вызовом метода `load`.

Изменение URL-адреса дистанционной вкладки

Метод `url` позволяет изменить URL-адрес, используемый для получения содержимого дистанционной вкладки. Аргументами этого метода являются индекс интересующей нас вкладки и требуемый новый URL-адрес. Соответствующий пример приведен в листинге 20.17.

Листинг 20.17. Использование метода `url` для изменения источника содержимого дистанционной вкладки

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $('#tabs').tabs({
            add: function(event, ui) {
                $(this).tabs("load", ui.index);
            },
            load: function(event, ui) {
                $(this).tabs("select", ui.index);
            }
        });

        $('<button id=change>Изменить URL</button>')
            .appendTo("#dc1");

        $('button').button().click(function(e) {
                var tabsElem = $('#tabs');
                switch (this.id) {
                    case "add":
                        tabsElem.tabs("add", "tabflowers.html",
                            $('#tabLabel').val());
                        break;
                    case "remove":
                        tabsElem.tabs("remove", tabsElem
                            .tabs("option", "selected"))
                        break;
                    case "change":
                        var selectedIndex = tabsElem
                            .tabs("option", "selected");
                        tabsElem.tabs("url", selectedIndex,
                            "tabflowers.html");
                        tabsElem.tabs("load", selectedIndex);
                        break;
                }
            })
        });
    </script>
...
```

В этом сценарии добавляется кнопка Изменить URL, щелчок на которой приводит к вызову метода `url` и изменению источника содержимого для текущей вкладки. В процессе изменения URL-адреса можно перейти от использования локального содержимого к дистанционному содержимому (обратный переход осуществить труднее, поскольку при этом потребуется связать вкладку с локальным элементом).

Если изменение URL-адреса осуществляется с целью создания дистанционной вкладки, то содержимое сможет загрузиться лишь после того, как пользователь отключит, а затем заново активизирует вкладку. Эта проблема решается путем вызова метода `load` для немедленной принудительной отправки Ajax-запроса.

Совет. В отношении описанного изменения URL-адреса действует правило ограничения домена. Подробнее об этом правиле говорится в главе 14.

Автоматический циклический показ вкладок

С помощью метода `rotate` можно организовать циклически повторяющееся отображение вкладок виджета. Это средство может быть весьма полезным, особенно если у вас имеется впечатляющее визуальное содержимое, которое пользователь иначе может и не увидеть (именно такой подход используется на туристическом

сайте, услугами которого я обычно пользуюсь при выборе места проведения своего отпуска, для отображения ярких картинок экзотических мест с помощью вкладок jQuery UI). Мой пример не будет столь же красочным, но его хватит для того, чтобы продемонстрировать суть используемого подхода. Соответствующий демонстрационный пример приведен в листинге 20.18.

Листинг 20.18. Автоматический циклический показ вкладок

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"},
                        {"name":"Примулы","product":"primula"},
                        {"name":"Подснежники",
                            "product":"snowdrop"},
                        {"name":"Гвоздики","product":"carnation"},
                        {"name":"Лилии","product":"lily"},
                        {"name":"Орхидеи","product":"orchid"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 3).appendTo("#tab1");
            elems.slice(3, 6).appendTo("#tab2");
            elems.slice(6).appendTo("#tab3");

            $('#tabs').tabs({
                fx: {
                    opacity: "toggle",
                    duration: "normal"
                }
            }).tabs("rotate", "5000", false);

            $('button').button();
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
```

```
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="tabs">
            <ul>
                <li><a href="#tab1">Ряд 1</a>
                <li><a href="#tab2">Ряд 2</a>
                <li><a href="#tab3">Ряд 3</a>
            </ul>
            <div id="tab1"></div>
            <div id="tab2"></div>
            <div id="tab3"></div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button></div>
    </form>
</body>
</html>
```

Аргументами метода `rotate` задаются количество миллисекунд, в течение которых должна отображаться каждая вкладка, и логическое значение, указывающее на то, должен ли продолжаться автоматический показ вкладок после того, как пользователь явным образом активизирует какую-либо вкладку.

Я рекомендую использовать медленные переходы между вкладками. Указанные в примере пять секунд — это разумный минимум. Ускорение этого процесса будет вызывать у пользователя дискомфорт. Уменьшение этого промежутка до двух секунд будет создавать стробоскопический эффект и явно не годится для наших целей. В этом примере я использую анимационный эффект, чтобы переход от одной вкладки к другой меньше раздражал пользователя. Средства анимации и эффекты, предлагаемые библиотекой jQuery, подробно рассмотрены в главе 10, а о дополнительных возможностях, появившихся в jQuery UI, будет рассказано в главе 34.

Возобновление циклического показа вкладок, прерванного действиями пользователя

Я придерживаюсь того мнения, что опция, управляющая возобновлением циклического показа вкладок после того, как пользователь активизировал одну из них, всегда должна быть установлена равной `false`. Раз уж пользователь выбрал вкладку для просмотра и активизировал ее, значит, принцип отображения содержимого ему понятен. Возобновляя циклический показ вкладок, вы рискуете заменить содержимое, выбранное пользователем, содержимым, которое его не интересует и от просмотра которого он отказался.

Если же вам действительно необходимо возвратиться к циклическому показу содержимого, то я рекомендую делать это несколько иначе. Проблемы, связанные с возобновлением показа вкладок с помощью метода `rotate`, частично обусловлены тем, что циклический показ вкладок в данном случае начинается по истечении заранее заданного фиксированного времени. Применительно к нашему примеру это означает, что для изучения содержимого активизированной вкладки у пользователя будет ровно пять секунд, потому что затем текущее содержимое будет заменено другим содержимым, не представляющим для пользователя никакого интереса. В листинге 20.19 продемонстрировано, как добиться более корректного поведения виджета по отношению к пользователю за счет использования метода `rotate` совместно с другими событиями виджета Tabs.

Листинг 20.19. Альтернативный способ возобновления циклического показа вкладок

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = [{"name":"Астры","product":"astor"},
                    {"name":"Нарциссы","product":"daffodil"},
                    {"name":"Розы","product":"rose"},
                    {"name":"Пионы","product":"peony"},
                    {"name":"Примулы","product":"primula"},
                    {"name":"Подснежники","product":"snowdrop"},
                    {"name":"Гвоздики","product":"carnation"},
                    {"name":"Лилии","product":"lily"},
                    {"name":"Орхидеи","product":"orchid"}];

        var elems = $('#flowerTmpl').tmpl(data);
        elems.slice(0, 3).appendTo("#tab1");
        elems.slice(3, 6).appendTo("#tab2");
        elems.slice(6).appendTo("#tab3");

        var displayDuration = 5000;
        var loadFactor = 5;
        var selectCount = 0;

        $('#tabs').tabs({
            fx: {
                opacity: "toggle",
                duration: "normal"
            },
            select: function() {
                var localCount = ++selectCount;
                setTimeout(function() {
                    if (localCount == selectCount) {
                        $('#tabs').tabs("rotate", displayDuration,
                            false)
                    }
                }, displayDuration * loadFactor)
            }
        }).tabs("rotate", displayDuration, false);

        $('button').button();
    });
</script>
...
```

В этом сценарии метод `rotate` используется для настройки циклического показа вкладок, который не возобновляется автоматически после того, как он был прерван действиями пользователя. Данный сценарий реагирует на событие `select`, которое происходит, когда вкладка активизируется пользователем (но не тогда, когда она автоматически выбирается в процессе циклического показа). Внутри обработчика этого события мы используем функцию `setTimeout()` для организации отложенного вызова другой функции. В этой функции метод `rotate` вызывается лишь в том случае, если с момента первоначальной реакции сценария на наступление события `select` пользователь не активизировал никакую другую вкладку.

Длительность паузы, по окончании которой становится возможным возобновление циклического показа вкладок, выбрана кратной длительности отображения одной вкладки. В нашем примере значение опции, определяющей длительность

паузы, составляет 5000, а величина множителя задана равной 5. Это означает, что каждая вкладка отображается в течение 5 секунд, а циклический показ будет возобновляться через 25 секунд после того, как пользователь в очередной раз активизирует какую-либо вкладку. Значение множителя необходимо выбирать в зависимости от характера представляемого содержимого, чтобы у пользователя был достаточный запас времени для ознакомления с содержимым активизированной им вкладки.

Предупреждение. Используйте циклический показ вкладок лишь в тех случаях, когда содержимое не требует взаимодействия с пользователем. Например, если вкладки применяются для отображения форм, то после того, как пользователь начнет вводить данные в элементы input, любая возможность возобновления автоматического показа вкладок должна быть исключена.

Использование событий виджета Tabs

Выше уже были приведены примеры работы с некоторыми из событий, поддерживаемых виджетом jQuery UI Tabs, а их полный перечень представлен в табл. 20.4.

Таблица 20.4. События виджета Tabs

Событие	*Описание*
create	Происходит, когда виджет Tabs применяется к базовому HTML-элементу
select	Происходит при активизации вкладки пользователем или при вызове метода select
load	Происходит по окончании загрузки содержимого для дистанционной вкладки
show	Происходит всякий раз, когда вкладка отображается для пользователя
add	Происходит при добавлении вкладки в виджет
remove	Происходит при удалении вкладки из виджета
enable	Происходит при включении функциональных возможностей вкладки
disable	Происходит при отключении вкладки

Событиям select, show, load и add предоставляется объект ui, для которого определены три свойства, содержащие полезную информацию о событии. Эти свойства описаны в табл. 20.5. События create, remove, enable и disable дополнительную информацию не предоставляют.

Таблица 20.5. Свойства объекта ui, предоставляющие информацию о событиях выбранных вкладок

Свойство	*Описание*
index	Индекс (номер) активной вкладки
panel	Объект HTMLElement, в котором находится содержимое активной вкладки
tab	URL-адрес источника содержимого для активной вкладки

Мы сосредоточим свое внимание на двух событиях, представляющих для нас наибольший интерес (select и show), а также на различиях между ними. Понимание сущности этих различий послужит основой для использования виджета Tabs в более сложных ситуациях. В этом разделе будет показано, как разбить HTML-форму на

части и отобразить ее отдельные разделы в виде вкладок, а затем использовать события select и show для проверки корректности заполнения формы при переходе от одной вкладки к другой.

Использование вкладок для отображения формы

Эта методика позволяет упростить работу с крупными формами, одновременно предоставляя пользователю возможность судить о том, как далеко он продвинулся в отношении заполнения формы данными. В качестве образца мы используем форму, представленную в листинге 20.20.

Листинг 20.20. Документ, содержащий форму

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #tab2 input, #tab3 input {width: 250px; text-align: left}
        #tab1, #tab2, #tab3 {padding: 10px}
        .fl {float: left}
        #buttonDiv {clear: both}
        #tabs, h1 {margin: 10px}
        .regLabel {width: auto}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 2).appendTo("#row1");
            elems.slice(2, 4).appendTo("#row2");

            var detailsData = [{name: "Имя", hint:
                "Введите ваше имя"},
                {name: "Улица", hint: "Введите название улицы"},
                {name: "Город", hint: "Введите название города"},
                {name: "Штат", hint: "Введите название штата"},
                {name: "Zip-код", hint: "Введите ваш ZIP-код"}];

            $('#detailsTmpl').tmpl(detailsData).appendTo("#tab2")
                .clone().appendTo("#tab3")

            $('button').button();
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell ui-widget">
```

```
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0"/>
        </div>
    </script>
    <script id="detailsTmpl" type="text/x-jquery-tmpl">
        <div class="ui-widget">
            <label for="${name}">${name}:</label>
            <input name="${name}"  placeholder="${hint}"/>
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="tabs" class="ui-widget">
            <ul>
                <li><a href="#tab1">1. Выберите товар</a>
                <li><a href="#tab2">2. Информация о вас</a>
                <li><a href="#tab3">3. Адрес доставки </a>
            </ul>

            <div id="tab1">
                <h2>1. Выберите товар</h2>
                <div id="row1"></div>
                <div id="row2"></div>
            </div>
            <div id="tab2" class="fl">
                <h2>2. Информация о вас</h2></div>
            <div id="tab3" class="fl">
                <h2>3. Адрес доставки</h2>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button></div>
    </form>
</body>
</html>
```

Чтобы несколько оживить предыдущие примеры, я добавил в этот документ дополнительное информационное наполнение и структуру. Ассортимент цветочной продукции уменьшился, но зато документ обогатился новыми разделами, позволяющими получать от пользователя его персональные данные и адрес доставки. Базовая форма этого документа представлена на рис. 20.13.

В этой форме нет ничего примечательного, но она хорошо подходит для использования вместе с виджетом Tabs jQuery UI, поскольку ее можно представить в виде независимых разделов, каждый из которых может отображаться на вкладке.

Все содержимое добавляется в документ программным путем с помощью подключаемого модуля шаблона данных и массивов JavaScript. Как нетрудно заметить, для генерации элементов из данных, их клонирования по мере необходимости и последующего добавления полученных результатов в документ используется функциональность jQuery, описанная в предыдущих главах. Применять вкладки для отображения форм вовсе не обязательно, но я считаю, что в книге, посвященной jQuery, следует стремиться к максимально полному использованию базовых средств.

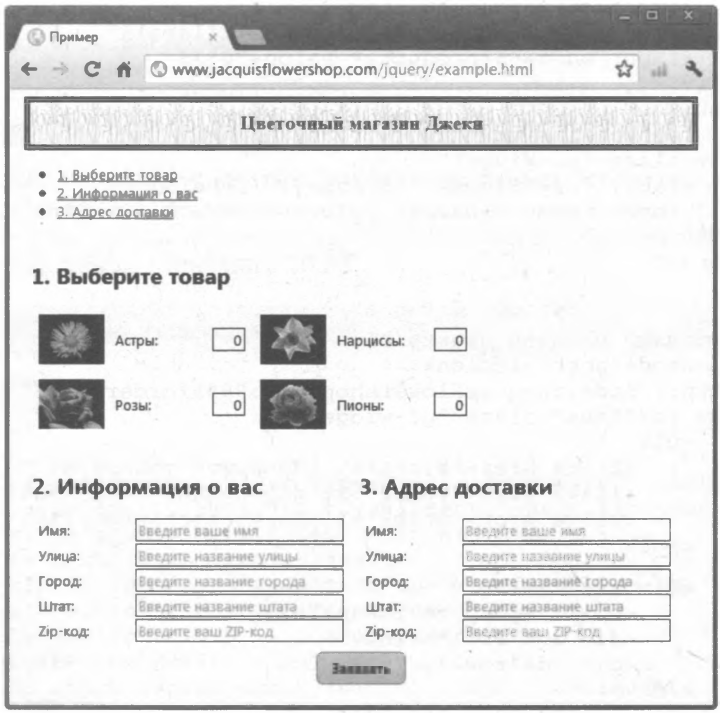


Рис. 20.13. Состоящая из нескольких разделов форма, предназначенная для использования с виджетом Tabs

Кроме того, на рисунке представлены элемент `ul` и содержащиеся в нем ссылки, указывающие на элементы содержимого. Обычно я скрываю эти элементы, но мне хотелось продемонстрировать полезный побочный эффект, привносимый структурой, которая используется виджетом Tabs для ярлыков. Поскольку здесь создается список, каждый элемент которого содержит ссылку, то щелчок на ссылке позволяет перейти к нужной части документа, а если ссылка указывает на другой файл, то браузер выполняет переход к указанному другому документу.

Применение вкладок

Теперь ничто не мешает нам создать виджет Tabs. В листинге 20.21 представлены изменения, которые необходимо внести для этого в элемент `script`. Никакие другие изменения не требуются.

Листинг 20.21. Создание виджета Tabs

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"}];

            var elems = $('#flowerTmpl').tmpl(data);
```

```
            elems.slice(0, 2).appendTo("#row1");
            elems.slice(2, 4).appendTo("#row2");

            var detailsData = [{name: "Имя", hint:
                "Введите ваше имя"},
                {name: "Улица", hint: "Введите название улицы"},
                {name: "Город", hint: "Введите название города"},
                {name: "Штат", hint: "Введите название штата"},
                {name: "Zip-код", hint: "Введите ваш ZIP-код"}];

            $('#detailsTmpl').tmpl(detailsData).appendTo("#tab2")
                .clone().appendTo("#tab3")

            $('.fl').removeClass("fl");
            $('#tabs').tabs().find("h2").remove();

            $('button').button();
        });
    </script>
...
```

Здесь удаляется класс `fl`, который использовался для размещения содержимого, представляющего сведения о покупателе и адресе доставки, а также элементы `h2`, которые использовались в качестве заголовков разделов. Далее вызывается метод `tabs()`, который использует контейнерные элементы в качестве основы для создания вкладок, как показано на рис. 20.14.

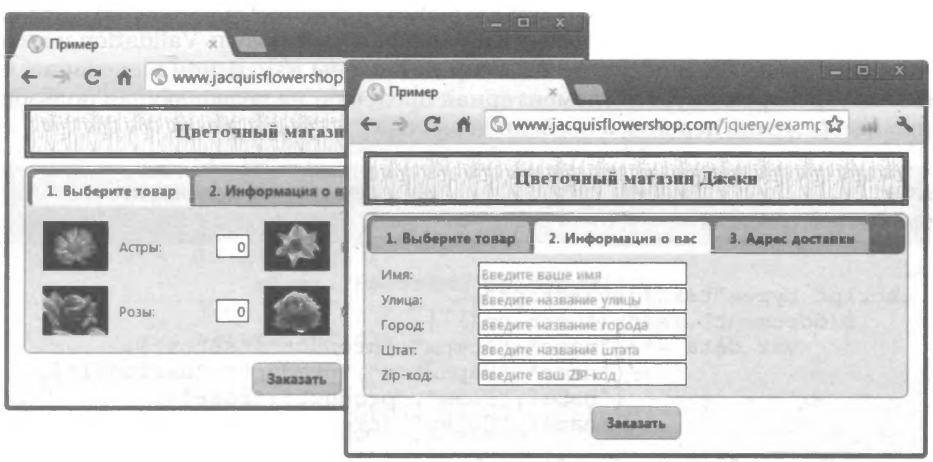


Рис. 20.14. Применение вкладок к форме

Обработка нажатий кнопки

Чтобы упростить заполнение полей формы за счет использования вкладок, зарегистрируем обработчик события `click()` для кнопки отправки формы. Действие по умолчанию, предусмотренное для данного события, отменяется в этом обработчике и заменяется переходом к следующей вкладке, пока не будет достигнута последняя вкладка. После этого щелчок на этой кнопке приводит к отправке формы на сервер. Соответствующие дополнения в сценарий приведены в листинге 20.22.

Листинг 20.22. Последовательное прохождение разделов формы с помощью кнопки отправки формы

```
...
$('button').button().click(function(e) {
    var tabsElem = $('#tabs');
    var activeTab = tabsElem.tabs("option", "selected");
    if (activeTab < (tabsElem.tabs("length") -1)) {
        e.preventDefault();
        tabsElem.tabs("select", activeTab + 1)
    }
});
...
```

Для определения того, является ли вкладка, с которой в данный момент работает пользователь, последней, используются опция `selected` и метод `length`, а для перехода к следующей вкладке — метод `select`. Метод `preventDefault()` вызывается лишь в том случае, если пользователь работает не с последней вкладкой, что обеспечивает отправку формы лишь в случае достижения пользователем последней вкладки.

Проверка данных формы

На данный момент пользователь может просто перейти к последней вкладке и отправить форму. Чтобы исключить эту возможность, применим элементарную проверку формы. Для простоты будем выполнять проверку вручную, но в реальных проектах для этого лучше использовать подключаемый модуль Validation и методы, описанные в главе 13. В листинге 20.23 представлены изменения в сценарии, с помощью которых реализуется элементарная проверка, не позволяющая пользователю преждевременно перейти к последней вкладке.

Листинг 20.23. Предотвращение пропуска пользователем вкладок с помощью элементарной проверки

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            var data = [{"name":"Астры","product":"astor"},
                        {"name":"Нарциссы","product":"daffodil"},
                        {"name":"Розы","product":"rose"},
                        {"name":"Пионы","product":"peony"}];

            var elems = $('#flowerTmpl').tmpl(data);
            elems.slice(0, 2).appendTo("#row1");
            elems.slice(2, 4).appendTo("#row2");

            var detailsData = [{name: "Имя", hint:
                "Введите ваше имя"},
                {name: "Улица", hint: "Введите название улицы"},
                {name: "Город", hint: "Введите название города"},
                {name: "Штат", hint: "Введите название штата"},
                {name: "Zip-код", hint: "Введите ваш ZIP-код"}];

            $('#detailsTmpl').tmpl(detailsData).appendTo("#tab2")
                .clone().appendTo("#tab3")
```

```
            var visiblePanel;
            var visibleIndex;

            $('.fl').removeClass("fl");
            $('#tabs').tabs({
                show: function(event, ui) {
                    visiblePanel = ui.panel;
                    visibleIndex = ui.index;
                },
                select: function(event, ui) {
                    if (ui.index > visibleIndex &&
                        !validateTab(visiblePanel)) {
                        event.preventDefault();
                    }
                }
            }).find("h2").remove();

            function validateTab(contentPanel) {
                var valid = false;
                if (contentPanel.id == "tab1") {
                    var productCount = 0;
                    $('#tab1 input').each(function(index, elem) {
                        productCount += Number($(elem).val());
                    })
                    valid = (productCount > 0);
                } else {
                    var emptyCount = 0;
                    $(contentPanel).find("input")
                        .each(function(index, elem) {
                        if ($(elem).val() == "") {
                            emptyCount++;
                        }
                    })
                    valid = (emptyCount == 0);
                }
                if (!valid) {
                    alert("Проблемы проверки!");
                }
                return valid;
            }

            $('button').button().click(function(e) {
                var tabsElem = $('#tabs');
                var activeTab = tabsElem.tabs("option",
                    "selected");
                if (activeTab < (tabsElem.tabs("length") -1)) {
                    e.preventDefault();
                    tabsElem.tabs("select", activeTab + 1)
                }
            });
        });
    </script>
...
```

Проблема, с которой вы сталкиваетесь, когда пытаетесь проверить часть формы, содержащуюся в одной из вкладок, состоит в том, что виджет Tabs не предлагает удобного способа определения того, какая из панелей содержимого активна.

Конечно, можно углубиться в код CSS-стилей или HTML-элементов, создаваемых виджетом, но при этом вы рискуете вновь столкнуться с проблемами, если команда разработчиков jQuery UI в какой-то момент внесет изменения в работу виджета.

Для отслеживания индекса активной вкладки и панели содержимого мы используем событие show. С этой целью мы определяем две переменные.

```
...
var visiblePanel;
var visibleIndex;
...
```

Значения этих переменных обновляются всякий раз, когда запускается событие show.

```
...
show: function(event, ui) {
    visiblePanel = ui.panel;
    visibleIndex = ui.index;
},
...
```

Затем мы используем событие select для вызова своего метода, осуществляющего проверку заполнения формы.

```
...
select: function(event, ui) {
    if (ui.index > visibleIndex &&
        !validateTab(visiblePanel)) {
        event.preventDefault();
    }
}
...
```

Для данной методики различия между событиями select и show играют существенную роль. Событие show запускается всякий раз, когда виджет Tabs отображает вкладку. Это может происходить в ответ на выполнение ввода пользователем, вызов метода select, первоначальное создание виджета или при использовании опции rotate. Событие show запускается и в тех случаях, когда отображение новой вкладки наступает как побочный эффект другой операции. Например, отключение активной вкладки запустит событие show, когда виджет Tabs отобразит для пользователя следующую доступную ему включенную вкладку. Событие select запустится лишь в том случае, если пользователь явно выберет вкладку или если будет вызван метод select.

Указанное различие позволяет отслеживать любую смену вкладок, используя событие show, и проверять пользовательский ввод лишь в ответ на наступление события select.

Обратите внимание на то, что проверка выполняется в функции-обработчике лишь при переходе пользователя к следующим вкладкам. Если событие select запускается для вкладки, индекс которой меньше индекса текущей отображаемой вкладки, то проверка не выполняется. Это позволяет пользователю вернуться к предыдущим вкладкам для изменения введенных данных.

Наконец, обратите внимание на то, как предотвращается преждевременный переход пользователя на следующую вкладку путем вызова метода preventDefault() для объекта Event, передаваемого функции-обработчику. Действием по умолчанию для события select является отображение выбранной вкладки, и отмена этого действия вынуждает виджет оставаться на текущей вкладке. Результат такой проверки показан на рис. 20.15. В данном примере проверка тривиальна, и в тех случаях, когда

не все поля оказываются заполненными, всего лишь выводится диалоговое окно с простым сообщением. В реальных проектах следует использовать подключаемый модуль Validation, описанный в главе 13.

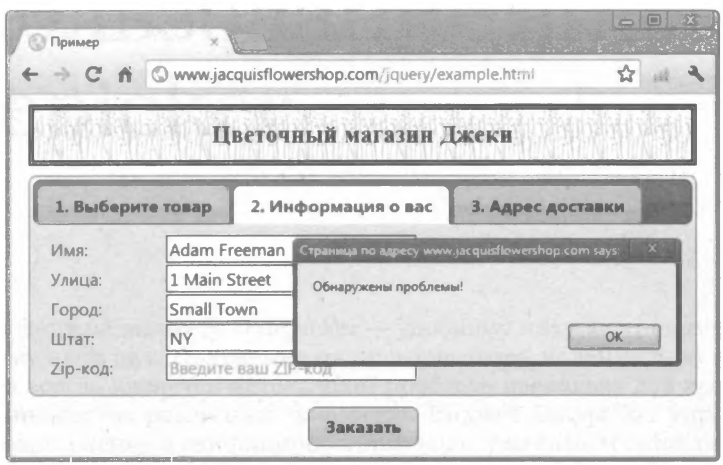


Рис. 20.15. Вывод диалогового окна с диагностическим сообщением в случае обнаружение ошибки в ходе проверки

Резюме

В этой главе вы познакомились с виджетом jQuery UI Tabs. Он отличается сложностью и обладает богатыми функциональными возможностями, что позволяет использовать его в самых разнообразных ситуациях. Я очень часто применяю его в своей практике. Он гибок и легко настраивается, а с самой идеей выборочного отображения содержимого путем предоставления его на отдельных вкладках пользователи, как правило, хорошо знакомы, чего нельзя сказать о других виджетах, таких как Accordion.

ГЛАВА 21

Использование виджета Datepicker

Эта глава посвящена виджету Datepicker — удобному интерактивному календарю, облегчающему ввод дат. Получение от пользователей календарной информации в виде текста всегда является источником проблем, поскольку для выражения дат существует множество различных форматов. Виджет Datepicker упрощает выбор даты и ее представление в унифицированном виде, тем самым снижая вероятность появления ошибок. Перечень тем, рассматриваемых в данной главе, приведен в табл. 21.1.

Таблица 21.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Создание всплывающего календаря jQuery UI	Используйте метод `datepicker()` для элемента `input`	1
Создание встроенного календаря	Используйте метод `datepicker()` для элемента `span` или `div`	2
Указание даты, отображаемой в календаре	Используйте опцию `defaultDate`	3
Указание дополнительного элемента, который будет обновляться при выборе даты пользователем	Используйте опцию `altField`	4
Изменение действия, инициирующего отображение календаря на экране	Используйте опцию `showOn`	5
Задание текста, отображаемого на кнопке запуска календаря	Используйте опцию `buttonText`	6
Вывод изображения вместо кнопки запуска	Используйте опции `buttonImage` и `buttonImageOnly`	7
Ограничение диапазона допустимых дат	Используйте опции `constrainInput`, `minDate` и `maxDate`	8, 9
Отображение нескольких месяцев в календаре	Используйте опцию `numberOfMonths`	10–12
Включение раскрывающихся списков для упрощения навигации по месяцам и годам	Используйте опции `changeMonth` и `changeYear`	13
Отображение информации о неделях в календаре	Используйте опции `showWeek` и `changeYear`	14

Окончание табл. 21.1

Задача	Решение	Листинг
Заполнение календарной сетки датами из предыдущего и последующего месяцев	Используйте опции `showOtherMonths` и `selectOtherMonths`	15
Отображение кнопочной панели в нижней части календаря	Используйте опции `showButtonBar` и `gotoCurrent`	16
Отображение подсказок с описанием формата дат для пользователя	Используйте опцию `appendText` (или заполнитель HTML5)	17, 18
Получение и установка даты программным способом	Используйте методы `getDate` и `setDate`	19
Отображение и сокрытие всплывающего календаря программным способом	Используйте события `show` и `hide`	20
Реагирование на переход пользователя к другому месяцу или году	Используйте событие `onChangeMonthYear`	21
Реагирование на закрытие всплывающего календаря	Используйте событие `close`	22
Локализация календаря	Используйте файл локализации календаря jQuery UI	23

Создание виджета Datepicker

Существуют два основных способа создания виджета Datepicker. Чаще всего его присоединяют к элементу `input` с помощью метода `datepicker()`. Немедленного изменения внешнего вида элемента при этом не происходит, но как только элемент получит фокус ввода (при выполнении щелчка на нем или переходе к нему от других элементов с помощью клавиши <Tab>), рядом с ним отобразится календарь, с помощью которого можно будет выбрать требуемую дату. Календари описанного типа называются *всплывающими* календарями. Пример создания такого календаря приведен в листинге 21.1[1].

Листинг 21.1. Создание всплывающего календаря

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input{width: 250px; text-align:left}
    </style>
```

[1] Вопросы языковой локализации виджета Datepicker обсуждаются в конце главы. Тривиальный способ решения этой проблемы состоит в ручном редактировании соответствующих фрагментов кода функции `Datepicker()` в файле `jquery-ui-1.8.16.custom.js`. — *Примеч. ред.*

```
    <script type="text/javascript">
        $(document).ready(function() {
            $('#datep').datepicker();
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div class="ui-widget">
            <label for="datep">Дата: </label><input id="datep" />
        </div>
    </form>
</body>
</html>
```

На рис. 21.1 показано, что происходит при перемещении фокуса в поле ввода.

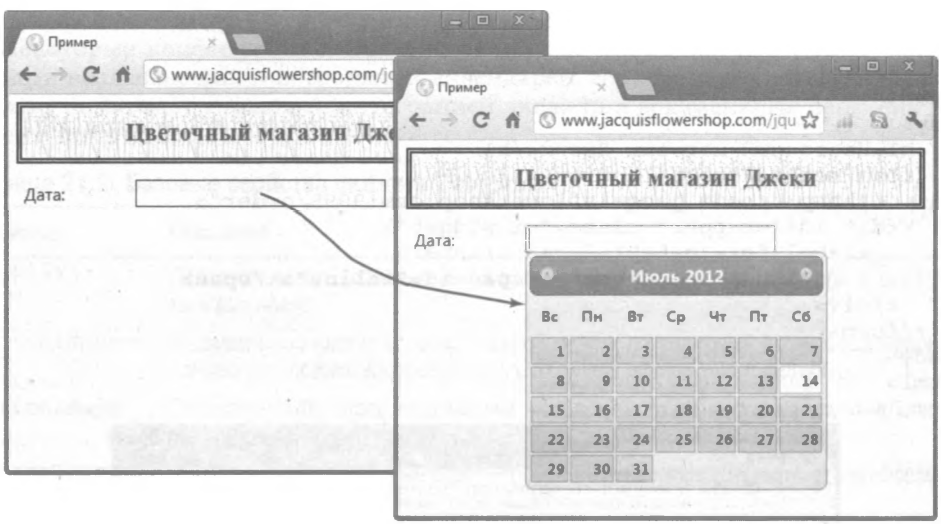


Рис. 21.1. Открытие всплывающего календаря Datepicker при получении элементом `input` *фокуса ввода*

Пользователь может либо ввести дату вручную, либо выбрать ее с помощью календаря. Как только элемент `input` потеряет фокус или пользователь нажмет клавишу <Enter> (или <Esc>), календарь исчезнет.

Создание встроенного календаря Datepicker

Второй способ использования виджета Datepicker предполагает его *встраивание* в документ. Для этого следует выбрать элемент `div` или `span` с помощью jQuery и вызвать метод `datepicker()`. Встроенный календарь отображается все время, пока виден HTML-элемент, на основе которого он создан. Пример создания встроенного календаря приведен в листинге 21.2.

В этом примере в качестве базового HTML-элемента для создания виджета Datepicker используется элемент `span`. Результат представлен на рис. 21.2.

Листинг 21.2. Создание встроенного календаря Datepicker

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 250px; text-align: left; margin-right: 10px}
        #wrapper > * {float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#inline').datepicker();
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
      <div id="wrapper" class="ui-widget">
        <label for="datep">Дата: </label>
            <input id="datep"/><span id="inline"></span>
      </div>
    </form>
</body>
</html>
```

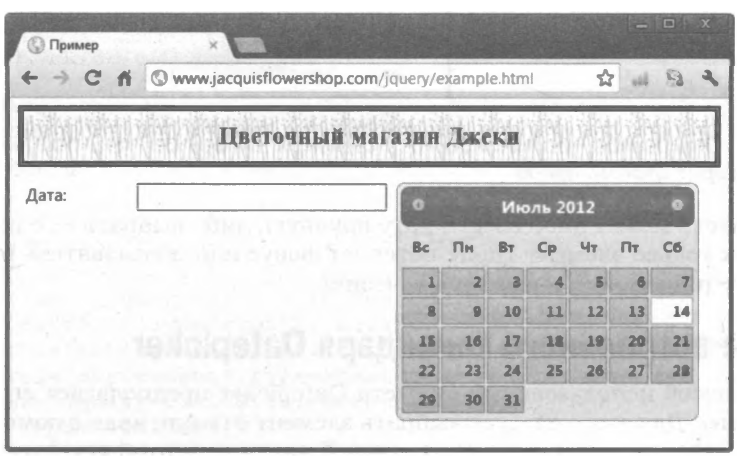


Рис. 21.2. Встроенный календарь Datepicker

Встроенный календарь может быть полезен, если вы не хотите работать с всплывающими объектами. Конечно, существуют приложения, в которых работа с датами ведется настолько интенсивно, что имеет смысл постоянно иметь календарь

под рукой. Но в большинстве случаев целесообразно оставлять календарь скрытым до тех пор, пока в нем не возникнет необходимость. Трудности с сокрытием и отображением встроенного виджета Datepicker обусловлены тем, что его добавление в документ влечет за собой необходимость изменения компоновки страницы, что может порождать проблемы представления. С моей точки зрения, в большинстве ситуаций гораздо удобнее использовать всплывающий виджет .

Настройка виджета Datepicker

Если до этого вам уже приходилось иметь дело с обработкой дат, то вам хорошо известно, насколько сложно работать с этой категорией данных. Отражением этой сложности является многочисленность свойств, поддерживаемых виджетом Datepicker. Описанию групп родственных свойств, с помощью которых настраивается данный виджет, посвящено несколько следующих разделов.

Базовые настройки

Некоторые конфигурационные опции используются для настройки базовых свойств всплывающих и встроенных календарей. Эти свойства очень важны, поскольку позволяют управлять интеграцией виджета в документ. Их перечень приведен в табл. 21.2.

Таблица 21.2. Базовые свойства виджета Datepicker

Свойство	*Описание*
`altField`	Позволяет задать дополнительное поле, которое будет обновляться при выборе даты в календаре
`buttonImageOnly`	Указывает, что вместо вспомогательной кнопки, позволяющей открыть календарь, должно использоваться изображение, заданное опцией `buttonImage`
`buttonImage`	Определяет URL-адрес изображения, используемого для вспомогательной кнопки открытия всплывающего календаря. По умолчанию не используется
`buttonText`	Определяет текст, который будет отображаться на кнопке открытия всплывающего календаря. Текстом по умолчанию является многоточие (. . .)
`defaultDate`	Позволяет установить дату, которая будет подсвечена при открытии календаря
`disabled`	Указывает, должен ли виджет быть первоначально отключен. Значение по умолчанию — `false`
`showOn`	Определяет действие, инициирующее открытие всплывающего календаря. Значение по умолчанию — `focus`

Указание даты, используемой по умолчанию

Простейшая настройка одновременно является и наиболее полезной. С помощью опции `defaultDate` можно указать дату, которая будет автоматически подсвечиваться при открытии календаря.

Если значение опции `defaultDate` не установлено, вместо него используется текущая дата. (Разумеется, здесь имеется в виду дата, установленная в пользовательской системе. Установка часовых поясов, переход на летнее время, неправильно настроенная конфигурация — все это может стать причиной того, что отображаемая для пользователя дата не будет совпадать с той, на которую вы рассчитываете.)

Совет. Эта опция используется лишь в случае отсутствия атрибута `value` в элементе `input`. Если этот атрибут присутствует (независимо от того, включен ли он в документ вами или появился в результате предварительного выбора пользователем), то виджет Datepicker будет использовать его значение `value`.

Если необходимо, чтобы календарь открывался с другой начальной датой, можно установить ее, воспользовавшись одним из способов, описанных в табл. 21.3.

Таблица 21.3. Возможные значения опции `defaultDate`

Значение	*Описание*
`null`	Используется текущая дата
Объект `Date`	Используется значение, представленное в виде JavaScript-объекта `Date`
+дни, *-дни*	Используется дата, отличающаяся от текущей даты на указанное количество дней. Так, `+3` означает дату, которая наступит через три дня после текущей, а `-2` — дату двухдневной давности
`+1d +7w -1m +1y`	Используется дата, которая отсчитывается от текущей даты и выражается посредством количества дней (`d`), недель (`w`), месяцев (`m`) и лет (`y`), определяющих величину сдвига даты вперед (`+`) или назад (`-`) по времени. Допускается смешивание положительных и отрицательных значений в одной дате. Например, комбинации значений `-1d +1m`, используемой совместно с датой `12 ноября 2011 года`, соответствует `11 декабря 2011`

Пример использования опции `defaultDate` для указания даты, которая наступит через пять лет, приведен в листинге 21.3.

Листинг 21.3. Использование опции `defaultDate`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep').datepicker({
            defaultDate: "+4m +5y"
        });
    });
</script>
...
```

Предположим, текущей дате соответствует июль 2012 года. Тогда, как показано на рис. 21.3, дате, определяемой значением опции `defaultDate`, соответствует ноябрь 2017 года.

Описанный формат указания относительных дат встретится вам еще не раз. Это очень гибкий формат, обеспечивающий необходимую точность. Точно так же, как это сделано в примере, можно опустить любой интервал, который не собираетесь изменять. Например, вместо значения `+0d +0w +4m +5y` вполне можно использовать значение `+4m +5y`. В этом формате удобно то, что он допускает смешивание положительных и отрицательных значений для различных интервалов, что позволяет точно определить нужную дату.

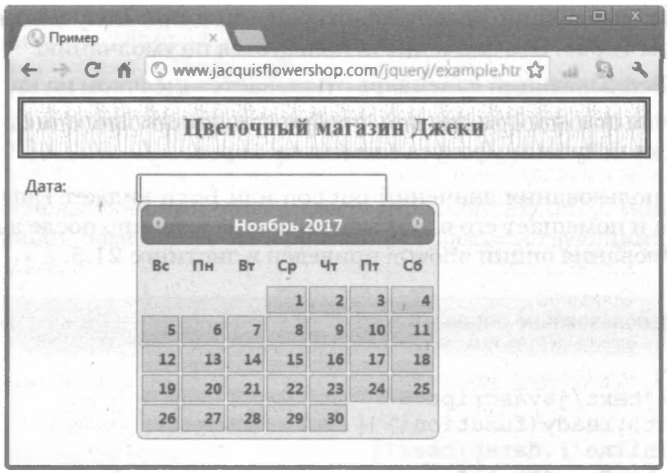


Рис. 21.3. Отображение в календаре будущей даты с использованием опции `defaultDate`

Определение дополнительного элемента

Опция `altField` позволяет определить дополнительный элемент `input`, который будет обновляться одновременно с выбором даты в календаре. Это позволяет очень просто организовать связь между встроенным календарем Datepicker и некоторым элементом, однако данная возможность будет полезной и в случае использования всплывающего календаря. Пример использования опции `altField` для отображения даты, выбранной во встроенном календаре, приведен в листинге 21.4.

Листинг 21.4. Использование опции `altField` во встроенномм календаре

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#inline').datepicker({
            altField: "#datep"
        });
    });
</script>
...
```

В этом примере для указания дополнительного элемента используется строка селектора, но в качестве значения опции `altField` можно использовать также объект `jQuery` или объект `HTMLElement`. Получаемый в этом примере результат состоит в том, что всякий раз, когда пользователь выбирает дату в календаре, она отображается в элементе `input`.

Определение события, инициирующего открытие всплывающего календаря

Опция `showOn` позволяет управлять событием, в ответ на которое должен отображаться всплывающий календарь. Эта опция может принимать одно из трех значений.

- `focus`. Всплывающий календарь открывается при получении фокуса ввода элементом `input`. Это значение используется по умолчанию.
- `button`. Всплывающий календарь открывается щелчком на кнопке.
- `both`. Всплывающий календарь отображается как после щелчка на кнопке, так и после получения фокуса элементом `input`.

В случае использования значений `button` или `both` виджет Datepicker создает элемент `button` и помещает его в документ непосредственно после элемента `input`. Пример использования опции `showOn` приведен в листинге 21.5.

Листинг 21.5. Использование опции `showOn`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#inline').datepicker({
            showOn: "#both"
        });
    });
</script>
...
```

Как показано на рис. 21.4, в документе появилась кнопка. Поскольку в этом примере опции `showOn` присвоено значение `both`, всплывающий календарь будет отображаться как при щелчке на кнопке, так и при получении фокуса элементом `input`.

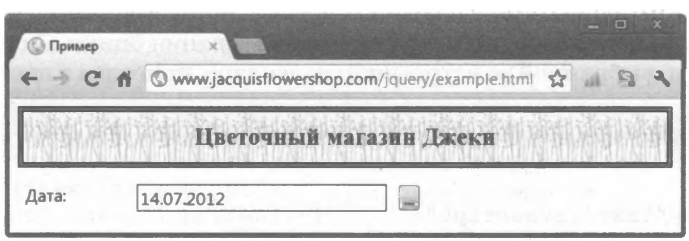


Рис. 21.4. Добавление кнопки в документ в ответ на действие, заданное опцией `showOn`

Совет. Кнопка, добавляемая виджетом Datepicker, не является виджетом Button jQuery UI. Если вы хотите, чтобы все кнопки были однотипными, выберите элемент `button` и вызовите метод `button()` jQuery UI, как описано в главе 18.

Элемент `button` можно стилизовать с помощью опций `buttonImage` и `buttonText`. Если задать в опции `buttonImage` URL-адрес изображения, то виджет Datepicker поместит это изображение на кнопку. Кроме того, связанный с кнопкой текст, заданный по умолчанию (многоточие), можно заменить другим тестком с помощью опции `buttonText`, как показано в листинге 21.6.

Листинг 21.6. Использование опции `buttonText`

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```
        $('#inline').datepicker({
            showOn: "#both",
            buttonText: "Выбрать"
        });
    });
</script>
...
```

Совместно используя опции `buttonImage` и `buttonTextOnly`, можно вообще избавиться от кнопки, заменив ее изображением. Соответствующий пример приведен в листинге 21.7.

Листинг 21.7. Использование изображения вместо кнопки

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input{width: 250px; text-align:left}
        #dpcontainer * {vertical-align: middle}
        #dpcontainer img {width: 35px;}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#datep').datepicker({
                showOn: "both",
                buttonImage: "right.png",
                buttonImageOnly: true
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
        <div id="dpcontainer" class="ui-widget">
            <label for="datep">Дата: </label><input id="datep" />
        </div>
    </form>
</body>
</html>
```

В этом примере задается изображение `right.png`, а для опции `buttonImageOnly` устанавливается значение `true`. Кроме того, в документ добавлено несколько CSS-стилей, управляющих размещением изображения относительно элементов `label` и `input`. Виджет Datepicker не может самостоятельно определить, куда именно следует поместить элемент `img`, и поэтому для правильного расположения этого элемента `img` в документе пришлось применить стили CSS. Результат использования изображения вместо кнопки представлен на рис. 21.5.

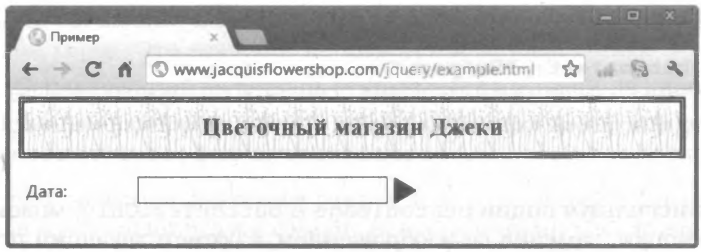


Рис. 21.5. Использование изображения вместо кнопки во всплывающем календаре

Управление выбором даты

Назначение виджета Datepicker — предоставить пользователю возможность выбрать дату, но во многих случаях в этот процесс требуется вводить некоторые ограничения. В табл. 21.4 описаны настройки, позволяющие устанавливать ограничения, вынуждающие пользователя выбирать лишь допустимые даты.

Таблица 21.4. Свойства виджета Datepicker, обеспечивающие управление выбором дат

Свойство	*Описание*
`changeMonth`	Если эта опция равна `true`, то в календаре отображается раскрывающийся список, обеспечивающий возможность непосредственной навигации по месяцам. Значение по умолчанию — `false`
`changeYear`	Если эта опция равна `true`, то в календаре отображается раскрывающийся список, обеспечивающий возможность непосредственной навигации по годам. Значение по умолчанию — `false`
`constrainInput`	Если эта опция равна `true`, то виджет Datepicker проверяет соответствие содержимого поля ввода заданному формату даты. Значение по умолчанию — `true`
`hideIfNoPrevNext`	Если эта опция равна `true`, то значки, позволяющие перемещаться по календарю вперед и назад относительно отображаемой даты, полностью скрываются, а не просто отключаются. Значение по умолчанию — `false`
`maxDate`	Определяет максимальную дату, доступную для выбора. По умолчанию это ограничение отсутствует
`minDate`	Определяет минимальную дату, доступную для выбора. По умолчанию это ограничение отсутствует
`numberOfMonths`	Определяет количество месяцев, одновременно отображаемых в календаре. Значение по умолчанию — `1`
`showCurrentAtPos`	Если для календаря задано одновременное отображение нескольких месяцев, то данная опция определяет номер позиции, в которой должен отображаться текущий или заданный по умолчанию месяц. Значение по умолчанию — `0`
`stepMonths`	Определяет, на сколько месяцев вперед или назад должна сдвигаться отображаемая в календаре дата при щелчке на кнопке перехода вперед или назад во времени
`yearRange`	Определяет диапазон лет, доступных для выбора в раскрывающемся списке, добавляемом с помощью опции `changeYear`. По умолчанию этот список включает десять предыдущих и десять последующих лет, а также текущий год

Ограничение вводимых символов и диапазона дат

Присвоив опции constrainInput значение true, можно ограничить ввод символов в текстовом поле лишь теми символами, которые соответствуют строго определенному формату. Допустимый набор символов определяется *настройками локализации*, о которых речь пойдет далее. Если локализация виджета Datepicker не выполнялась, то следует ожидать, что в набор допустимых символов будут входить лишь цифры и символ косой черты (/). Использование указанного значения опции constrainInput еще не означает, что пользователь не сможет ввести недопустимую дату, например 99/99/99, но способствует значительному уменьшению вероятности возникновения ошибок. Значение этой настройки еще более возрастает, если для опции showOn установлено значение button, поскольку в этом случае всплывающий календарь не будет автоматически открываться при получении фокуса полем ввода. Обычно пользователи охотно используют календарь для выбора даты, но для этого они должны его видеть. Если же календарь не отображается на экране, то далеко не каждый пользователь догадается, что для открытия календаря достаточно щелкнуть на кнопке. Любая возможность непосредственного ввода даты в текстовом поле, которую вы предоставляете пользователю, увеличивает вероятность того, что формат введенной им даты окажется неверным. Пример использования опции constrainInput приведен в листинге 21.8.

Листинг 21.8. Применение базовых ограничений при выборе даты

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 250px; text-align: left; margin-right: 10px}
        #wrapper > * {float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#datep').datepicker({
                constrainInput: true,
                minDate: "-3",
                maxDate: "+1"
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
      <div id="wrapper" class="ui-widget">
        <label for="datep">Дата: </label>
            <input id="datep"/><span id="inline"></span>
      </div>
```

```
    </form>
</body>
</html>
```

Здесь значение `true` присваивается опции `constrainInput` лишь для наглядности, поскольку оно является значением по умолчанию для этой опции. Остальные две опции позволяют ограничить диапазон дат доступных для выбора, минимальной и максимальной датой. Как и в случае опции `defaultDate`, которая рассматривалась ранее, в качестве значений опций `minDate` и `maxDate` могут использоваться `null` (дата не определена), JavaScript-объект `Date`, число дней и строка относительной даты. В данном примере выбрано числовое представление, указывающее количество дней, отсчитываемых относительно текущей даты. На рис. 21.6 видно, что Datepicker отключает ячейки календаря, недоступные для выбора.

Рис. 21.6. Ограничение дат, доступных для выбора

Совет. Обратите внимание, что кнопки листания месяцев автоматически отключаются, если в них нет необходимости. Эти кнопки располагаются слева и справа вверху календаря и позволяют быстро переходить к следующему или предыдущему месяцам. Как показано на рис. 21.6, все доступные для выбора даты относятся к текущему месяцу, и поэтому обе кнопки отключены. В подобных ситуациях эти кнопки при необходимости можно полностью скрыть, установив для опции `hideIfNoPrevNext` значение `true`.

Значение `minDate` не обязательно должно относиться к прошлому, а значение `maxDate` — к будущему, равно как вовсе не обязательно задавать одновременно оба значения. Чтобы предоставить пользователю возможность выбирать даты из диапазона, которому предшествует некий "подготовительный" период, можно присвоить опции `minDate` значение, относящееся к будущему, и тем самым лишить пользователя возможности выбора дат, относящихся к упомянутому периоду, как показано в листинге 21.9.

Листинг 21.9. Наложение одностороннего ограничения на выбор дат в календаре для создания "периода задержки"

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep').datepicker({
            minDate: "+7",
        });
```

```
    });
</script>
...
```

В этом примере мы указали, что пользователю не должны быть доступны для выбора даты, предшествующие той, которая наступит через неделю после текущей даты. Значение опции `maxDate` не определено, и это означает, что пользователь сможет выбрать любую дату, следующую за указанным "периодом задержки". Результат представлен на рис. 21.7. Обратите внимание, что кнопка перехода к следующему месяцу в данной ситуации включена, тогда как кнопка перехода к предыдущему месяцу отключена (ввиду отсутствия дат, относящихся к прошлому, которые были бы доступны пользователю для выбора).

Совет. Опции `minDate` и `maxDate` работают в сочетании с опцией `defaultDate`, откуда следует, что привязка диапазонов к текущей дате не является обязательной.

Рис. 21.7. Создание открытого диапазона выбора дат

Создание календаря, отображающего одновременно несколько месяцев

Виджет Datepicker позволяет установить количество месяцев, которые должны одновременно отображаться в календаре, посредством опции `numberOfMonths`. Это можно сделать, указав либо требуемое количество месяцев, либо массив из двух элементов, определяющий размеры месячной календарной сетки. В листинге 21.10 реализован подход, основанный на использовании массивов, который я считаю наиболее приспособленным для встроенных виджетов Datepicker, поскольку для всплывающих виджетов размер сетки часто оказывается слишком большим (к обсуждению этого вопроса мы еще вернемся).

Листинг 21.10. Использование опции `numberOfMonths`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#inline').datepicker({
            numberOfMonths: [1, 3]
        });
    });
</script>
...
```

В этом примере задана календарная сетка, высота которой соответствует одному месяцу, а ширина — трем. Результат представлен на рис. 21.8.

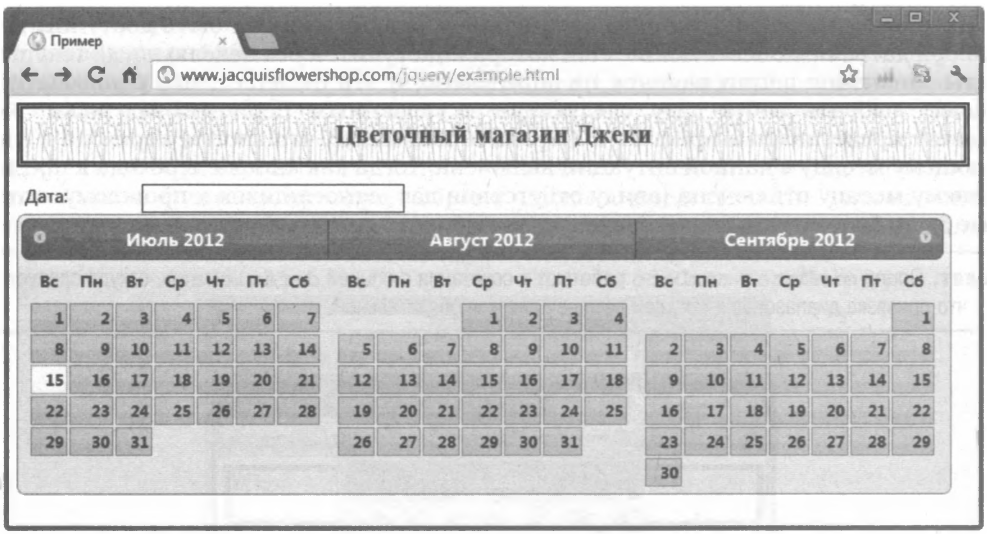


Рис. 21.8. Календарная сетка

Совет. Массив из двух элементов `[1, 3]` эквивалентен числовому значению `3`. Если значение опции `numberOfMonths` задано в виде числа, то виджет отображает соответствующее количество месяцев в одном ряду.

Причина, по которой я редко использую описанный подход, когда работаю с всплывающими календарями, заключается в том, что в случае больших сеток приходится делать определенные предположения относительно размеров окна браузера и дисплея. Окно всплывающего виджета Datepicker не является диалоговым окном операционной системы. Оно представляет собой искусно стилизованный HTML-элемент, который отображается как часть HTML-документа. Это означает, что при отображении крупных календарных сеток на небольшом экране или в маленьком окне браузера значительная часть остальной информации, содержащейся в документе, окажется смещенной за пределы экрана. Пример использования сетки месяцев во всплывающем виджете Datepicker приведен в листинге 21.11.

Листинг 21.11. Использование опции `numberOfMonths` во всплывающем виджете Datepicker

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep').datepicker({
            numberOfMonths: [1, 3]
        });
    });
</script>
...
```

Результат представлен на рис. 21.9. Вы видите, что не только многие из доступных дат оказались скрытыми от пользователя, но и кнопка для листания месяцев вперед вышла за пределы окна браузера.

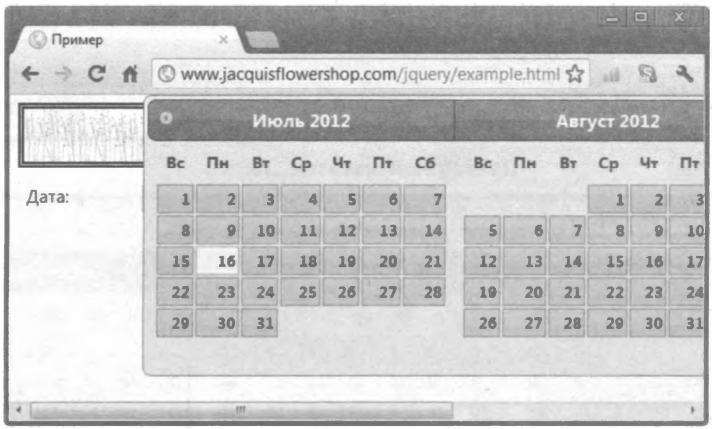


Рис. 21.9. Всплывающий виджет с крупной календарной сеткой

Позицию выбранной даты в календаре, отображающем одновременно несколько месяцев, можно изменить с помощью опции `showCurrentAtPos`. На рис. 21.9 показано, что по умолчанию первым отображается текущий месяц, вслед за которым идут два последующих месяца. Опция `showCurrentAtPos` принимает значение в виде индекса, отсчитываемого от нуля, который определяет номер позиции для отображения текущего месяца. Эта возможность очень удобна, если требуется, чтобы пользователь имел возможность выбирать даты, относящиеся как к прошлому, так и к будущему времени относительно текущей даты. Пример использования этой опции приведен в листинге 21.12.

Листинг 21.12. Использование опции `showCurrentAtPos`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#inline').datepicker({
            numberOfMonths: 3,

            showCurrentAtPos: 1
        });
    });
</script>
...
```

Здесь указано, что текущей дате должен соответствовать средний из отображаемых в календаре месяцев. Результат представлен на рис. 21.10.

Предоставление прямого доступа к месяцам и годам

Вместо того чтобы просто отображать месяцы и годы в заголовке календаря, можно обеспечить прямой доступ к соответствующим периодам времени с помощью раскрывающихся списков. Эта возможность позволяет ускорить работу в тех случаях, когда пользователям приходится выбирать даты в широком диапазоне

возможных значений, и реализуется с помощью опций `changeMonth` и `changeYear`. Присвоение значения `true` любой из этих опций активизирует соответствующий раскрывающийся список, причем допускается независимое управление активизацией этих списков. Пример использования этих опций приведен в листинге 21.13.

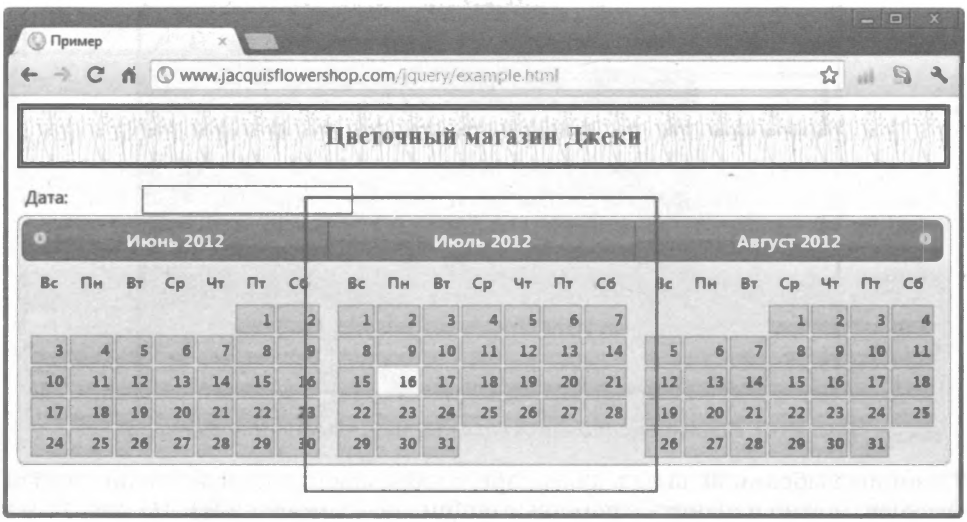


Рис. 21.10. Указание относительного местоположения текущего месяца в календарной сетке, отображающей несколько месяцев

Листинг 21.13. Предоставление прямого доступа к месяцам и годам с помощью раскрывающихся списков

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep').datepicker({
            changeMonth: true,
            changeYear: true,
            yearRange: "-1:+2"
        });
    });
</script>
...
```

В этом сценарии активизированы оба раскрывающихся списка. Кроме того, для ограничения диапазона лет, между которыми пользователю разрешены переходы, используется опция `yearRange`. Требуемый диапазон указан с помощью значения `-1:+2`, которое означает, что в календаре пользователю разрешены переходы в пределах одного года назад и двух лет вперед относительно текущего года. Раскрывающиеся списки и заданный диапазон лет отображены на рис. 21.11.

Совет. При определении годовых диапазонов с помощью опции `yearRange` можно указывать фактические годы. Так, в данном примере можно было бы указать значение `2011:2014`.

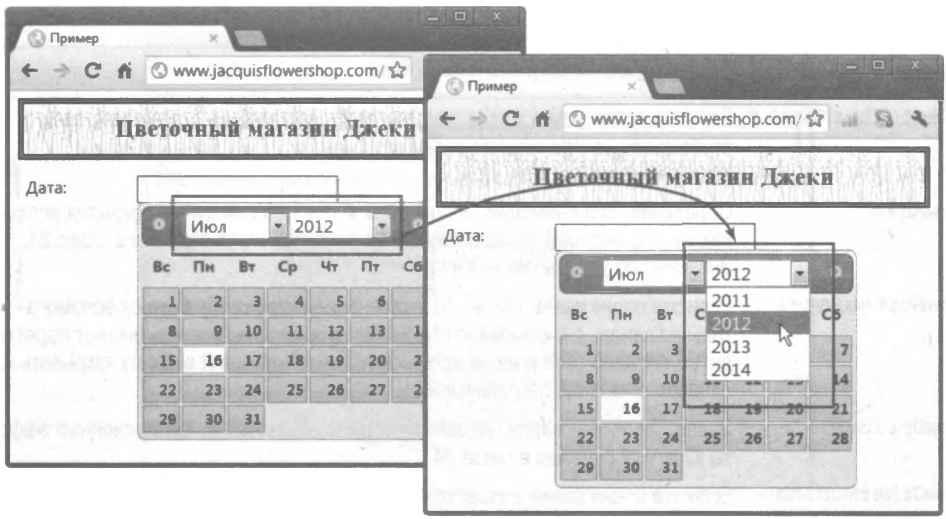


Рис. 21.11. Предоставление пользователю прямого доступа к годам и месяцам в календаре

Управление внешним видом виджета Datepicker

Существует целый ряд свойств, позволяющих настраивать внешний вид виджета Datepicker при его отображении для пользователя. Для выполнения обычных задач, связанных с выбором даты, можно удовлетвориться внешним видом календаря, используемым по умолчанию, с которым вы уже знакомы по предыдущим примерам, однако возможность настройки внешнего вида в соответствии с задачами конкретного веб-приложения чрезвычайно полезна. Свойства, обеспечивающие возможность изменения внешнего вида виджета Datepicker, приведены в табл. 21.5.

Таблица 21.5. Свойства, позволяющие управлять внешним видом виджета Datepicker

Свойство	*Описание*
`appendText`	Задает текст подсказки, содержащий дополнительную информацию, например пояснения относительно форматирования даты, который будет вставлен в документ после элемента `input`
`closeText`	Задает текст для кнопки закрытия календаря, отображающейся на панели вспомогательных кнопок, если она включена. Значение по умолчанию — `Закрыть`
`currentText`	Задает текст для кнопки возврата к текущему месяцу, отображающейся на панели вспомогательных кнопок, если она включена. Значение по умолчанию — `Сегодня`
`duration`	Задает скорость или длительность выполнения анимации, заданной опцией `showAnim`. Значение по умолчанию — `normal`. Анимационные эффекты jQuery UI описаны в главе 34
`gotoCurrent`	Если эта опция равна `true`, то кнопка `Сегодня`, находящаяся на панели вспомогательных кнопок, если она включена, будет осуществлять возврат к выбранной, а не к текущей дате. Значение по умолчанию — `false`

Окончание табл. 21.5

Свойство	Описание
selectOtherMonths	Если эта опция равна true, то становятся доступными для выбора даты, отображаемые в результате установки равным true значения опции showOtherMonths
showAnim	Определяет тип анимации, используемой для отображения и сокрытия всплывающих календарей. Анимационные эффекты jQuery UI описаны в главе 34. Значение по умолчанию — false
showButtonPanel	Если эта опция равна true, то в календаре будет отображаться вспомогательная панель с кнопками, с помощью которых пользователь сможет переходить к текущей дате и (если используется всплывающий виджет) закрывать календарь. Значение по умолчанию — false
showOptions	Задает опции анимации, указанной опцией showAnim. Анимационные эффекты jQuery UI описаны в главе 34
showOtherMonths	Если эта опция равна true, то пустые поля в календарной сетке будут заполняться датами из предыдущих и последующих месяцев. Значение по умолчанию — false
showWeek	Если значение этой опции равно true, то в календаре будет отображаться столбец с номерами недель. Значение по умолчанию — false
weekHeader	Задает заголовок столбца календаря с номерами недель, включенной с помощью опции showWeek. Значение по умолчанию — Нед

Отображение недель

В некоторых приложениях важно знать номер недели в году, на которую приходится та или иная дата. Например, это требуется во многих программах бухгалтерского учета. В виджете Datepicker управление отображением информации о неделях осуществляется с помощью опций showWeek и weekHeader, как показано в листинге 21.14.

Листинг 21.14. Отображение информации о неделях в виджете Datepicker

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep').datepicker({
            showWeek: true,
            weekHeader: "Неделя"
        });
    });
</script>
...
```

Если опция showWeek имеет значение true, то в календаре отображается столбец с номерами недель. По умолчанию заголовком этого столбца является Нед, однако его можно изменить с помощью опции weekHeader. В примере включено отображение столбца недель, название которого изменено на Неделя (рис. 21.12).

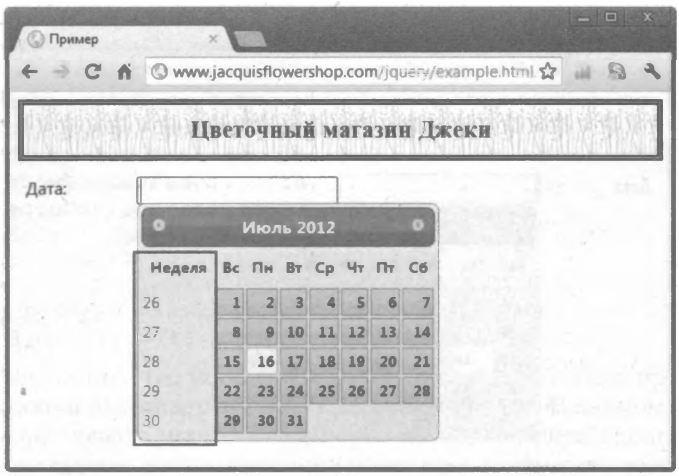


Рис. 21.12. Отображение информации о неделях в виджете Datepicker

Заполнение пустых ячеек календаря датами соседних месяцев

По умолчанию в календаре отображаются лишь даты, относящиеся к текущему месяцу. Это означает, что в календарной сетке могут присутствовать пустые ячейки. Можно разрешить заполнение пустых ячеек датами предыдущего и последующего месяцев, установив значение опции `showOtherMonths` равным `true`, как показано в листинге 21.15.

Листинг 21.15. Заполнение пустых ячеек календаря датами соседних месяцев

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep').datepicker({
            showOtherMonths: true
        });
    });
</script>
...
```

Результат представлен на рис. 21.13. В этом случае даты, относящиеся к другим месяцам, становятся доступными, если для опции `selectOtherMonths` установлено значение `true`.

Использование вспомогательной панели для дополнительных кнопок

Установка значения `true` для опции `showButtonBar` приводит к добавлению панели дополнительных кнопок, располагающейся в нижней части окна виджета Datepicker. В случае всплывающего календаря панель содержит две кнопки: Сегодня и Закрыть. Кнопка Сегодня позволяет вернуться к текущей дате, а кнопка Закрыть предназначена для закрытия окна календаря. Эти кнопки показаны на рис. 21.14. На панели встроенного календаря отображается только кнопка Сегодня.

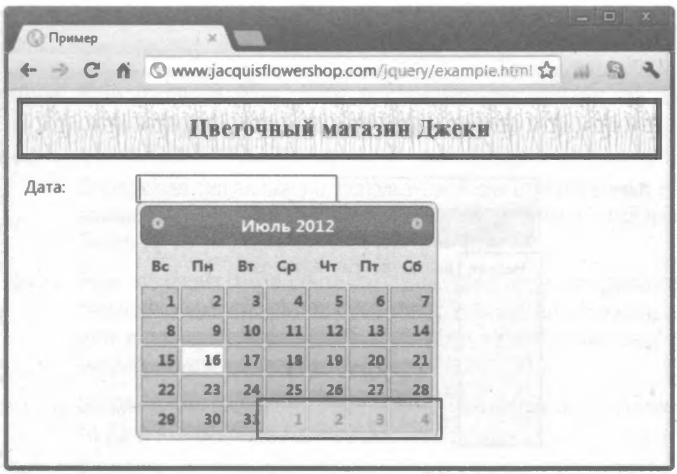


Рис. 21.13. Отображение дат, относящихся к предыдущему и последующему месяцам

Совет. Текст, используемый для кнопок Сегодня и Закрыть, можно изменить с помощью опций `currentText` и `closeText`.

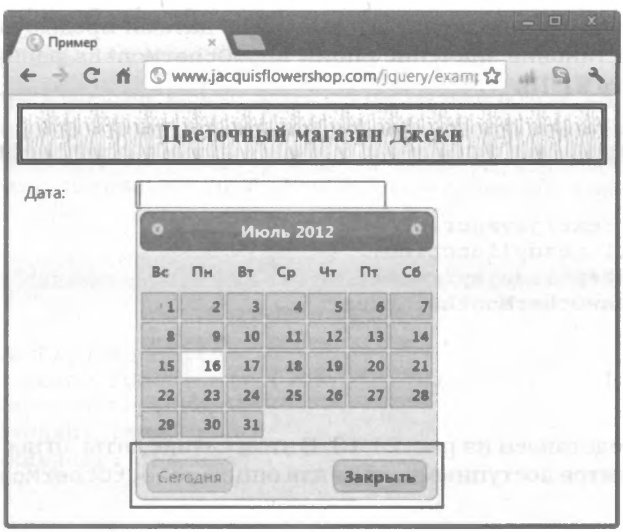


Рис. 21.14. Отображение вспомогательной панели дополнительных кнопок

Если для опции `goToCurrent` установить значение `true`, то календарь будет возвращаться к выбранной дате, а не текущей. Это средство удобно использовать, если виджет Datepicker сконфигурирован с определенным значением опции `defaultDate`. Если цель выбора даты связана с прошлыми или будущими событиями, то возврат к текущей дате не всегда целесообразен. Соответствующий пример приведен в листинге 21.16.

Листинг 21.16. Использование опции `goToCurrent`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep').datepicker({
            showButtonPanel: true,
            gotoCurrent: true,
            defaultDate: "+1m +1y"
        }).val("26.12.2012");
    });
</script>
...
```

Обратите внимание, что использование опции `goToCurrent` приводит к тому, что нажатие кнопки вызывает переход к выбранной дате. В данном примере такой выбранной датой служит значение атрибута `value` элемента `input`, но если пользователь выберет другую дату, а затем вновь откроет календарь, то нажатие кнопки будет возвращать календарь к дате, выбранной пользователем, а не к той, которая указана вами.

Предоставление пользователю информации о текущем формате указания дат

С помощью опции `appendText` можно предоставить пользователю дополнительную информацию об используемом формате указания дат. Соответствующий пример приведен в листинге 21.17.

Листинг 21.17. Использование опции `appendText` для предоставления информации о формате указания дат

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep).datepicker({
            appendText: '(dd.mm.yy)'
        });
    });
</script>
...
```

Виджет вставит указанный вами текст в документ, как показано на рис. 21.15.

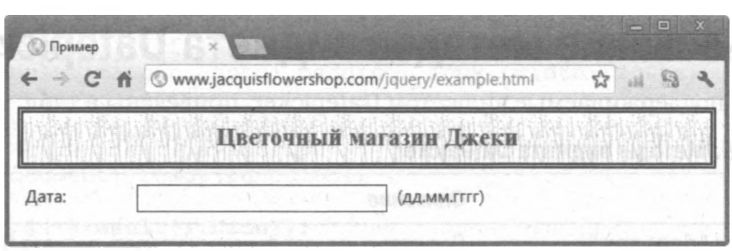


Рис. 21.15. Использование опции `appendText` для предоставления пользователю информации о формате указания дат

Эта опция весьма полезна в тех случаях, когда используется всплывающий календарь, открываемый кнопкой. Если пользователь может ввести дату, не прибегая к помощи календаря, то наличие подсказки об используемом формате дат может значительно уменьшить вероятность возникновения ошибок, требующих дополнительной обработки (что будет благом для вас и избавит пользователей от лишних хлопот).

Другой возможный подход, к которому я с недавних пор прибегаю как к более изящной альтернативе опции `appendText` виджета Datepicker, состоит в использовании предложенного в спецификации HTML5 атрибута `placeholder` элемента `input`. Соответствующий пример приведен в листинге 21.18.

Листинг 21.18. Предоставление информации о форматировании дат с помощью атрибута `placeholder`, определенного в спецификации HTML5

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#datep).attr("placeholder", "дд.мм.гггг").datepicker();
    });
</script>
...
```

Совершенно очевидно, что применимость такого подхода целиком зависит от того, поддерживает ли браузер пользователя спецификацию HTML5, но если это так, то результат выглядит очень элегантно. Я предпочитаю использовать именно этот вариант, поскольку он обеспечивает более тесную связь подсказки с текстовым полем ввода и не требует выделения дополнительного пространства в макете документа. Как выглядит такая подсказка в окне браузера Google Chrome, показано на рис. 21.16.

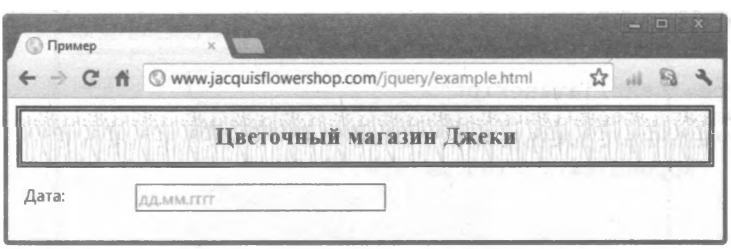


Рис. 21.16. Использование атрибута `placeholder` HTML5 для вывода подсказки об используемом формате даты

Использование методов виджета Datepicker

Методы, поддерживаемые виджетом Datepicker, приведены в табл. 21.6.

Таблица 21.6. Методы виджета Datepicker

Метод	Описание
`datepicker("destroy")`	Полностью удаляет функциональность виджета Datepicker из базового элемента
`datepicker("disable")`	Приостанавливает работу виджета Datepicker для базового элемента

Окончание табл. 21.6

Метод	Описание
`datepicker("enable")`	Возобновляет работу ранее приостановленного виджета Datepicker для базового элемента
`datepicker("option", опции)`	Позволяет получить или установить значения одной или нескольких опций виджета Datepicker
`datepicker("isDisabled")`	Возвращает `true`, если виджет Datepicker отключен
`datepicker("hide")`	Скрывает всплывающий календарь, если он видимый
`datepicker("show")`	Отображает всплывающий календарь, если он невидимый
`datepicker("refresh")`	Обновляет календарь для отражения выполненных в базовом элементе изменений
`datepicker("getDate")`	Получает дату, выбранную в календаре
`datepicker("setDate", дата)`	Устанавливает указанное значение в качестве выбранной даты календаря

Получение и изменение даты программным путем

Чаще всего я прибегаю к использованию методов `getDate` и `setDate` в тех случаях, когда хочу предоставить пользователям возможность выбора целых диапазонов дат с помощью нескольких виджетов Datepicker. В подобных ситуациях я предпочитаю не отображать выбранные даты в текстовых полях и вывожу лишь количество дней между заданными граничными датами. Соответствующий пример приведен в листинге 21.19.

Листинг 21.19. Использование двух календарей для выбора диапазона дат

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 250px; text-align: left; margin-right: 10px}
        #wrapper > * {float: left}
        #result {margin: auto; padding: 10px; width: 200px;
            clear: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#result').hide();

            $('#dateStart, #dateEnd').datepicker({
                minDate: "-7d",
                maxDate: "+7d",
```

```
                onSelect: function(date, datepicker) {
                    if (datepicker.id == "dateStart") {
                        $('#dateEnd').datepicker("setDate", date)
                            .datepicker("enable")
                            .datepicker("option", "minDate", date)
                    }

                    if (!$('#dateEnd').datepicker("isDisabled")) {
                        var startDate = $('#dateStart')
                            .datepicker("getDate");
                        var endDate = $('#dateEnd')
                            .datepicker("getDate");
                        var diff = endDate.getDate() -
                             startDate.getDate();
                        $('#dayCount').text(diff).parent().show();
                    }
                }

            }).filter("#dateEnd").datepicker("disable");
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
      <div id="wrapper" class="ui-widget">
        <label for="dateStart">Начало: </label>
            <span id="dateStart"></span>
        <label for="dateEnd">Конец: </label>
            <span id="dateEnd"></span>
      </div>
      <div id="result" class="ui-widget">
        Количество дней: <span id="dayCount"></span>
      </div>
    </form>
</body>
</html>
```

В этом примере создаются два календаря, один из которых сразу после загрузки документа находится в отключенном состоянии. Для реагирования на выбор дат пользователем используется событие `onSelect` (о котором мы подробнее поговорим далее). Как только пользователь выбирает дату в первом календаре, мы применяем метод `setDate` для подготовки второго календаря, а метод `getDate` — для получения дат из обоих календарей и последующего вычисления разницы в днях между первой и второй датами (для простоты предполагается, что обе даты относятся к одному и тому же месяцу). Вид документа в окне браузера представлен на рис. 21.17.

Отображение и сокрытие всплывающих календарей программным способом

Методы `show` и `hide` позволяют управлять выводом всплывающего календаря в окне программы. Это может пригодиться, если вы хотите, чтобы отображение календаря на экране связывалось не только с получением фокуса элементом `input` или со щелчком на кнопке, созданной виджетом Datepicker. Я не являюсь большим

сторонником создания кнопок в документе средствами Datepicker и поэтому время от времени использую указанные методы для управления календарем с помощью кнопки, которую создаю сам, как показано в листинге 21.20.

Рис. 21.17. Использование методов `getDate` *и* `setDate`

Листинг 21.20. Использование методов `show` и `hide`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 250px; text-align: left; margin-right: 10px}
        #wrapper > * {float: left}
        label {padding: 4px; text-align: right; width: auto}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#datep').datepicker();

             $('button').click(function(e) {
                e.preventDefault();
                $('#datep').datepicker("show");
                setTimeout(function() {
                    $('#datep').datepicker("hide");
                }, 5000)
            })

        });
```

```
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
      <div id="wrapper" class="ui-widget">
        <label for="datep">Дата: </label>
            <input id="datep"/><span id="inline"></span>
        <button>Datepicker</button>
      </div>
    </form>
</body>
</html>
```

При щелчке на кнопке вызывается метод `show`. Я не часто использую метод `hide`, поскольку предпочитаю, чтобы пользователь мог сам закрывать календарь, сделав окончательный выбор даты, но в данном случае для завершения примера используется функция `setTimeout()`, в результате чего календарь автоматически закрывается спустя пять секунд после нажатия кнопки.

Использование событий виджета Datepicker

Как и все виджеты jQuery UI, виджет Datepicker поддерживает набор событий, позволяющих получать уведомления о важных изменениях в состоянии приложения. Эти события приведены в табл. 21.7.

Таблица 21.7. События виджета Datepicker

Событие	*Описание*
`create`	Происходит при создании экземпляра Datepicker
`onChangeMonthYear`	Происходит, когда пользователь переходит к другому месяцу или году в календаре
`onClose`	Происходит при закрытии всплывающего календаря
`onSelect`	Происходит, когда пользователь выбирает дату

Я не буду повторно демонстрировать, как работает метод `onSelect`, поскольку он уже использовался в паре примеров, включая тот, который был рассмотрен в предыдущем разделе. Единственное, на что я хочу обратить ваше внимание, — это то, что аргументами, передаваемыми обработчику события, служат строковые представления выбранной даты и экземпляра Datepicker, породившего данное событие.

Реагирование на изменение месяца или года в календаре

Событие `onChangeMonth` позволяет реагировать на событие, которое происходит при выборе пользователем другого месяца или года, будь то с помощью раскрывающихся списков, включенных посредством использования опций `changeMonth` и `changeYear` или кнопок листания месяцев. Пример того, как данное событие можно использовать для синхронизации двух календарей, приведен в листинге 21.21.

Листинг 21.21. Использование события onChangeMonthYear

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 250px; text-align: left; margin-right: 10px}
        #wrapper > * {float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#dateStart, #dateEnd').datepicker({
                onSelect: function(date, datepicker) {
                    if (datepicker.id == "dateStart") {
                        $('#dateEnd').datepicker("setDate", date)
                    }
                },
                onChangeMonthYear: function(year, month,
                    datepicker) {
                    if (datepicker.id == "dateStart") {
                        var newDate = new Date();
                        newDate.setMonth(month -1);
                        newDate.setYear(year);
                        $('#dateEnd')
                            .datepicker("setDate", newDate);
                    }
                }
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
      <div id="wrapper" class="ui-widget">
        <label for="dateStart">Начало: </label>
            <span id="dateStart"></span>
        <label for="dateEnd">Конец: </label>
            <span id="dateEnd"></span>
      </div>
    </form>
</body>
</html>
```

В этом примере обработчик события onChangeMonthYear принимает три аргумента: год и месяц, отображаемые в календаре, а также экземпляр виджета Datepicker, являющийся источником события. Когда пользователь переходит к дру-

гому месяцу или году в первом календаре, для второго календаря устанавливается дата, обеспечивающая синхронизацию обоих календарей.

Обратите внимание на то, что в виджете Datepicker январю соответствует индекс 1, тогда как в объекте JavaScript Date этому месяцу соответствует индекс 0. Именно поэтому в процессе создания даты, которая должна отображаться во втором календаре, потребовалась следующая коррекция:

```
newDate.setMonth(month -1);
```

Реагирование на закрытие всплывающего календаря

Используя метод onClose, можно реагировать на закрытие всплывающего календаря. Это событие запускается даже в том случае, если пользователем не была выбрана дата в календаре. Аргументами обработчика события являются строковое представление даты (или пустая строка, если пользователь закрыл окно, не сделав выбора) и экземпляр Datepicker, породивший данное событие. Пример простых ответных действий на это событие представлен в листинге 21.22.

Листинг 21.22. Использование события onClose

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 250px; text-align: left; margin-right: 10px}
        #wrapper > * {float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
             $('#datep').datepicker({
                onClose: function(date, datepicker) {
                    if (date != "") {
                        alert("Выбранная дата: " + date);
                    }
                }
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
      <div id="wrapper" class="ui-widget">
        <label for="datep">Дата: </label><input id="datep"/>
      </div>
    </form>
</body>
</html>
```

В этом примере в случае выбора даты отображается диалоговое окно с сообщением о выбранной дате. Должен признаться, что я еще ни разу не использовал это событие в реальных проектах, тогда как событие `onSelect` считаю одним из наиболее полезных.

Локализация виджета Datepicker

В виджете jQuery UI Datepicker обеспечена довольно исчерпывающая поддержка различных стандартов форматирования дат, используемых по всему миру, которая включает 61 вариант локализации. Для получения доступа к ним необходимо импортировать в документ один вспомогательный сценарий JavaScript и указать виджету Datepicker, какой вариант локализации должен использоваться. Соответствующий пример приведен в листинге 21.23.

Листинг 21.23. Использование локализованного варианта виджета Datepicker

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script src="jquery-ui-i18n.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 250px; text-align: left; margin-right: 10px}
        #wrapper > * {float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#inline').datepicker($.datepicker.regional["uk"]);
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post" action=
        "http://node.jacquisflowershop.com:9999/order">
      <div id="wrapper" class="ui-widget">
        <label for="datep">Дата: </label>
            <input id="datep"/><span id="inline"></span>
      </div>
    </form>
</body>
</html>
```

Файл `jquery-ui-i18n.js` находится в папке `development-bundle/ui/i18n` архивного файла библиотеки jQuery UI, который вы ранее загрузили (см. главу 17). Скопируйте его в ту же папку, в которой находятся файлы сценариев основной библиотеки jQuery и библиотеки jQuery UI, и добавьте его в документ с помощью следующей строки.

```
<script src="jquery-ui-i18n.js"
    type="text/javascript"></script>
```

Нужный вариант локализации календаря можно указать на стадии создания виджета Datepicker.

```
$('#inline').datepicker($.datepicker.regional["uk"]);
```

Несмотря на некоторую запутанность синтаксиса этой инструкции, она позволяет задать требуемый вариант локализации. В данном выбран вариант локализации uk, который соответствует форматам дат, принятым в Украине. Результат представлен на рис. 21.18.

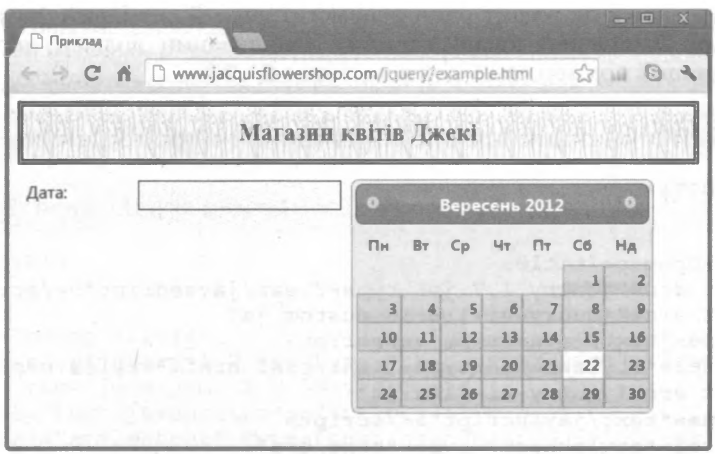


Рис. 21.18. Локализация календаря

Совет. Если нужные вам форматы дат в настоящее время не поддерживаются в jQuery UI, можно создать собственные варианты локализации. Более подробную информацию можно получить по следующему адресу:

`http://docs.jquery.com/UI/Datepicker/Localization`

Мои рекомендации относительно локализации сводятся к тому, что это надо либо делать, как следует, либо вообще не делать. Локализация, понимаемая в широком смысле, не может ограничиваться лишь форматированием дат и должна предоставлять пользователю интерфейс, который в полной мере учитывал бы такие аспекты, как используемый язык, географическое положение пользователя, а также региональные соглашения о форматах времени, валюты и других характеристик. Частичная локализация веб-приложений или непоследовательность в поддержке стандартов могут послужить причиной недовольства пользователей. Серьезный подход к решению проблем локализации приложений требует обращения к специалисту или компании, специализирующимся на выполнении такого рода работы. В этой области очень трудно избежать ошибок, и без помощи профессионалов вы будете обречены на неудачу.

Если у вас возникает соблазн локализовать приложение с помощью таких средств, как, например, Google Translate (с чем нередко приходится сталкиваться), то я рекомендую отказаться от этой идеи и использовать в своем приложении лишь поддержку вариантов локализации US English и US. Это ограничит вашу пользова-

тельскую базу клиентами, владеющими английским языком, для которых привычны соглашения относительно дат, валюты и т.п., принятые в США, но поможет избежать катастрофы, которая почти несомненно наступит в случае решения проблем локализации доморощенными методами.

Резюме

В этой главе была продемонстрирована работа виджета Datepicker jQuery UI, облегчающего выбор дат с помощью календаря. Этот виджет обладает гибкими свойствами, обеспечивающими возможность настройки его внешнего вида и способа выбора дат в соответствии с конкретными запросами. Мой собственный опыт работы с виджетом Datepicker говорит о том, что это средство оказывает неоценимую помощь в разрешении проблем форматирования, с которыми приходится сталкиваться при получении от пользователей информации, связанной с вводом дат.

ГЛАВА 22

Использование виджета Dialog

Виджет Dialog создает плавающее окно с заголовком и областью содержимого, внешне напоминающее диалоговые окна обычных приложений. Виджеты Dialog можно использовать как для вывода сообщений, так и для привлечения внимания пользователей к важным событиям. Однако, как и в случае любых других элементов, заслоняющих собой часть содержимого документа, в использовании диалоговых окон следует соблюдать умеренность и прибегать к их помощи лишь в ситуациях, когда вывод их содержимого в макете самого документа вызывает затруднения. Перечень тем, рассматриваемых в данной главе, приведен в табл. 22.1.

Таблица 22.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Создание виджета Dialog jQuery UI	Выберите элемент `div` с атрибутом `title` и вызовите метод `dialog()`	1
Отмена отображения диалогового окна сразу после его создания	Установите для опции `autoOpen` значение `false`	2
Предотвращение изменения размеров диалогового окна пользователем	Установите для опции `resizable` значение `false`	3
Изменение начальной позиции диалогового окна	Используйте опцию `position`	4
Добавление одной или нескольких кнопок в диалоговое окно	Используйте опцию `buttons`	5
Предотвращение перетаскивания диалогового окна пользователем или перемещения его в стек диалоговых окон	Используйте опции `draggable` и `stack`	6
Создание модального диалогового окна	Установите для опции `modal` значение `true`	7, 8
Программное открытие и закрытие диалогового окна	Используйте методы `open`, `close` и `isOpen`	9
Предотвращение закрытия диалогового окна	Возвратите значение `false` в функции — обработчике события `beforeClose`	10
Реагирование на перемещение или изменение размера диалогового окна пользователем	Организуйте обработку событий `dragStart`, `dragStop`, `drag`, `resizeStart`, `resizeStop` и `resize`	11

Создание виджета Dialog

Для создания виджета Dialog, позволяющего превращать блочные элементы (обычно — `div`) в диалоговые окна, следует выбрать элемент с помощью jQuery и вызвать для него метод `dialog()`. Виджет Dialog относится к числу тех виджетов, для нормальной работы которых требуется определенная структура HTML-элементов, хотя эта структура намного проще, чем, например, та, которая требуется для виджета Tabs. Документ, содержащий необходимые элементы и сценарий для создания виджета Dialog, представлен в листинге 22.1.

Листинг 22.1. Создание диалогового окна с помощью виджета Dialog

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#dialog').dialog();
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dialog" title="Диалоговое окно">
        Содержимое, которое будет отображаться в диалоговом окне.
        К этому <b>содержимому</b> могут применяться
        <em>стили</em>.
    </div>
</body>
</html>
```

Для виджета Dialog требуется элемент `div` с атрибутом `title`. Значение этого атрибута служит заголовком диалогового окна. Содержимое элемента `div` используется в качестве информации, отображаемой в диалоговом окне, и, как показывает пример, может включать в себя другие элементы. Вызов метода `dialog()` без передачи ему каких-либо опций, как это сделано в листинге, приводит к немедленному открытию диалогового окна. Вид диалогового окна в окне браузера приведен на рис. 22.1.

Диалоговое окно создается исключительно за счет использования специфической структуры HTML-элементов, а не средствами операционной системы. Это означает, что диалоговые окна jQuery UI ведут себя не совсем так, как обычные диалоговые окна. Они не будут видны при отображении пользователем всех открытых окон на рабочем столе, и путем изменения размера окна браузера можно добиться того, что какая-то часть диалогового окна jQuery UI (или все окно целиком) будет скрыта.

Команда jQuery UI хорошо поработала над тем, чтобы наделить диалоговое окно максимально широким набором различных возможностей. Пользователь может переместить диалоговое окно в пределах окна браузера, перетаскивая его заголо-

вок. Можно изменить размер диалогового окна с помощью кнопки-манипулятора, находящейся в его правом нижнем углу, или закрыть его, щелкнув на кнопке закрытия в правом верхнем углу. А поскольку виджет Dialog построен из HTML-элементов, его внешний вид определяется темой оформления jQuery UI, выбранной вами в главе 13, и он может включать в себя сложное HTML-содержимое, оформленное с помощью стилей.

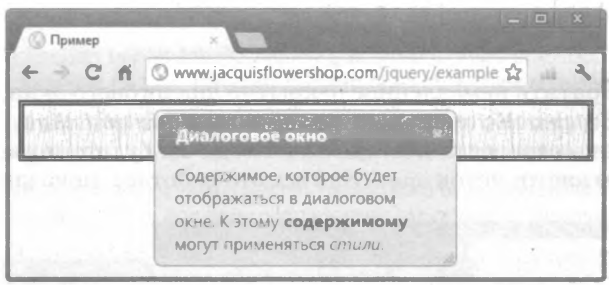


Рис. 22.1. Простое диалоговое окно

Прежде чем перейти к подробному рассмотрению свойств, методов и событий, поддерживаемых данным виджетом, хочу привести типичный пример его использования. Если метод `dialog()` вызывается без аргументов, то диалоговое окно открывается сразу же после того, как этот метод закончит свою работу. Как правило, это не очень удобно. В большинстве случаев вам будет требоваться диалоговое окно, которое создается после загрузки документа (но так, чтобы структура элементов оставалась невидимой для пользователей) и которое впоследствии можно отобразить в ответ на некоторое событие. Пример того, как это можно сделать, приведен в листинге 22.2.

Листинг 22.2. Отсроченное отображение диалогового окна jQuery UI

```
<!DOCTYPE html>

<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#dialog').dialog({
                autoOpen: false
            });

            $('button').button().click(function(e) {
                $('#dialog').dialog("open")
            })
        });
    </script>
</head>
```

```
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dialog" title="Диалоговое окно">
        Содержимое, которое будет отображаться в диалоговом окне.
        К этому <b>содержимому</b> могут применяться
        <em>стили</em>.
    </div>
    <button>Показать диалоговое окно</button>
</body>
</html>
```

Чтобы предотвратить немедленное открытие диалогового окна, следует использовать опцию `autoOpen`. Если значение этой опции равно `false`, то вновь созданное диалоговое окно останется невидимым. Когда вы будете готовы к тому, чтобы отобразить его, вызовите метод `open`. Как все это работает, показано на рис. 22.2.

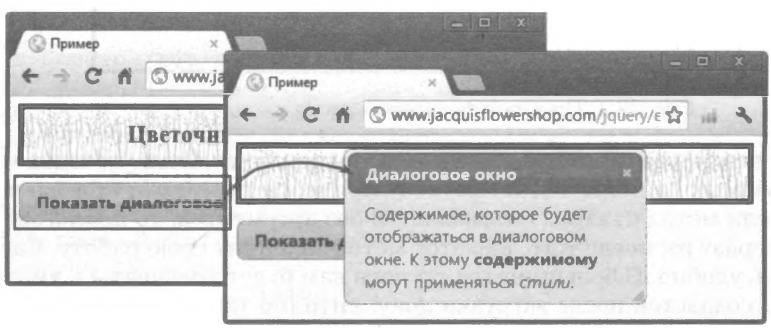


Рис. 22.2. Отсроченное отображение диалогового окна

Настройка виджета Dialog

Виджет Dialog поддерживает ряд свойств, обеспечивающих настройку конфигурации диалогового окна, представляемого пользователю, в соответствии с конкретными потребностями. С одним из них, свойством `autoOpen`, вы уже познакомились в предыдущем разделе, а их полный перечень приведен в табл. 22.2.

Таблица 22.2. Свойства виджета Dialog

Свойство	Описание
`autoOpen`	Если эта опция равна `true`, то диалоговое окно открывается сразу же после его создания с помощью метода `dialog()`. Значение по умолчанию — `true`
`buttons`	Позволяет указать набор кнопок, которые должны быть добавлены в виджет, и функции, которые будут вызываться после щелчка на соответствующей кнопке. По умолчанию кнопки отсутствуют
`closeOnEscape`	Если эта опция равна `true`, то диалоговое окно можно будет убрать с экрана, нажав клавишу <Esc>. Значение по умолчанию — `true`
`draggable`	Если эта опция равна `true`, то пользователь сможет перемещать диалоговое окно, перетаскивая его заголовок, в пределах окна браузера
`height`	Определяет начальную высоту диалогового окна в пикселях. По умолчанию имеет значение `auto`, при котором высота диалогового окна устанавливается автоматически

Окончание табл. 22.2

Свойство	Описание
hide	Определяет тип анимации, используемой для сокрытия диалогового окна. Эффекты jQuery UI описаны в главе 34
maxHeight	Определяет максимальную высоту диалогового окна в пикселях. По умолчанию имеет значение false, которому соответствует отсутствие ограничений по высоте
maxWidth	Определяет максимальную ширину диалогового окна в пикселях. По умолчанию имеет значение false, которому соответствует отсутствие ограничений по ширине
minHeight	Определяет минимальную высоту диалогового окна в пикселях. По умолчанию имеет значение false, которому соответствует отсутствие ограничений по высоте
minWidth	Определяет минимальную ширину диалогового окна в пикселях. По умолчанию имеет значение false, которому соответствует отсутствие ограничений по ширине
modal	Если эта опция равна true, то диалоговое окно будет создано как модальное, и пока оно не будет скрыто, пользователь не сможет взаимодействовать с документом
position	Определяет начальную позицию диалогового окна. Значение по умолчанию — center, которому соответствует расположение диалогового окна по центру окна браузера
resizable	Если эта опция равна true, то диалоговое окно будет иметь кнопку-манипулятор, с помощью которой пользователь сможет изменить его размер. Значение по умолчанию — true
show	Определяет тип анимации, используемой для отображения диалогового окна. Эффекты jQuery UI описаны в главе 34
stack	Если эта опция равна true, то щелчок на диалоговом окне перемещает его на передний план на экране. Значение по умолчанию — true
title	Определяет заголовок диалогового окна
width	Определяет начальную ширину диалогового окна в пикселях. По умолчанию имеет значение auto, при котором высота диалогового окна устанавливается автоматически

Настройка внешнего вида базового диалогового окна

Опция title позволяет создать диалоговое окно на основе элемента div, не имеющего атрибута title. Это может быть полезным, если у вас отсутствует возможность управлять генерацией элементов, которые вы хотите использовать в диалоговом окне. Пример использования опции title приведен в листинге 22.3.

Листинг 22.3. Использование опции title

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
```

```
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#dialog').dialog({
                title: "Привет",
                resizable: false
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dialog"
        Содержимое, которое будет отображаться в диалоговом окне.
        К этому <b>содержимому</b> могут применяться
        <em>стили</em>.
    </div>
</body>
</html>
```

В этом примере также была применена опция `resizable`. Она управляет наличием кнопки-манипулятора в правом нижнем углу диалогового окна. Лично мне больше нравятся диалоговые окна, не содержащие никаких лишних элементов, но обычно я оставляю значение опции `resizable` равным `true`, поскольку предпочитаю предоставлять пользователю возможность изменять размер диалогового окна в зависимости от размера содержимого. Вид диалогового окна в окне браузера представлен на рис. 22.3.

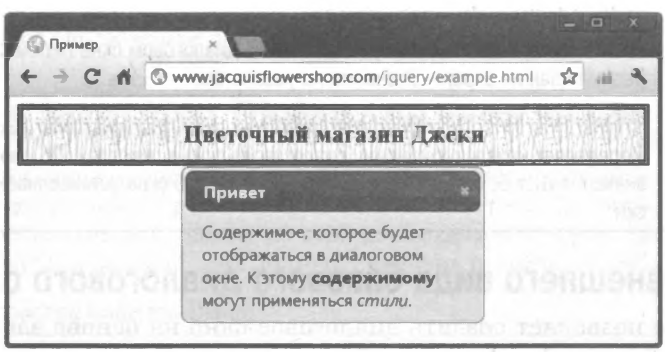


Рис. 22.3. Диалоговое окно с настраиваемым заголовком, не имеющее кнопки-манипулятора

Настройка местоположения диалогового окна

С помощью опции `position` можно указать, где именно в окне браузера будет отображено диалоговое окно. Для этого можно использовать три различных типа значений, описанных в табл. 22.3.

По умолчанию диалоговое окно располагается по центру окна браузера (значение `center`). Обычно эта позиция является наилучшей, однако для сравнения в листинге 22.4 приведен пример, в котором диалоговое окно позиционируется с использованием строковых значений.

Таблица 22.3. Типы значений, используемых в опции `position`

Значение	*Описание*
строка	Одно из следующих строковых значений: `center`, `left`, `right`, `top`, `bottom`
[число, число]	Массив из двух элементов, содержащий координаты `x` и `y` левого верхнего угла окна, например `[10, 20]`
[строка, строка]	Массив из двух элементов, содержащий строковые значения координат из приведенного выше списка, например `['left',top']`

Листинг 22.4. Позиционирование диалогового окна

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#dialog').dialog({
            title: "Позиционированное диалоговое окно",
            position: ["left", "top"]
        });
    });
</script>
...
```

Обратите внимание на то, что при использовании строковых значений первой указывается позиция вдоль оси x. Такой порядок указания координат отличается от того порядка их указания, который обычно используют в устной речи, описывая положение элементов. Как правило, в подобных ситуациях мне приходится делать над собой усилие, поскольку позиция, которую я мысленно определяю как *top left*, должна записываться в виде `[left, top]`. Конечный результат представлен на рис. 22.4.

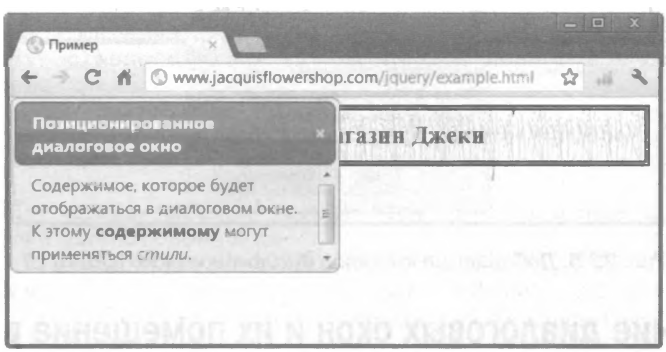


Рис. 22.4. Диалоговое окно, располагающееся в указанной позиции

Добавление кнопок в диалоговое окно

В диалоговое окно jQuery UI можно добавить кнопки, используя опцию `buttons`. Значением этой опции является массив объектов, у каждого из которых имеются свойства `text` и `click`. Значение свойства `text` используется в качестве надписи на кнопке, а значением свойства `click` определяется функция, которая будет вызываться при выполнении на кнопке щелчка. Пример использования опции `buttons` приведен в листинге 22.5.

Листинг 22.5. Добавление кнопок в диалоговое окно

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#dialog').dialog({
            title: "Диалоговое окно",
            buttons: [{text: "OK", click: function() {
            /* выполнить некоторые действия */}},
                      {text: "Отменить", click: function() {
                          $(this).dialog("close")}}]
        });
    });
</script>
...
```

В этом сценарии добавляются две кнопки. Функция для кнопки OK не выполняет никаких полезных действий, тогда как щелчок на кнопке Отменить приводит к закрытию диалогового окна. Обратите внимание на использование переменной `this` в селекторе jQuery внутри функции — обработчика события `click` для кнопки Отменить. Эта переменная указывает на элемент `div`, который использовался для создания диалогового окна. Внешний вид диалогового окна с добавленными кнопками представлен на рис. 22.5. В этом примере используется метод `close`, при вызове которого диалоговое окно исчезает с экрана. Методы виджета Dialog описываются более подробно далее.

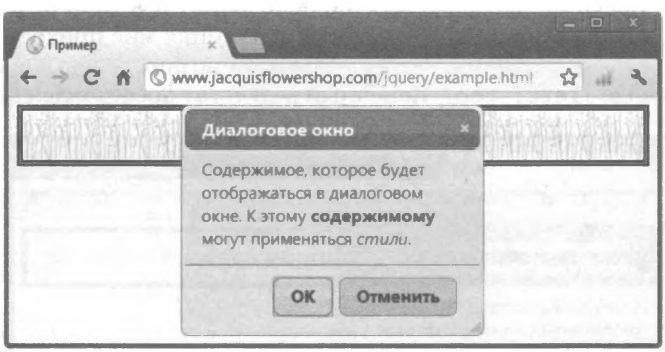


Рис. 22.5. Добавление кнопок в диалоговое окно jQuery UI

Перемещение диалоговых окон и их помещение в стек

Опция `draggable` определяет, сможет ли пользователь перемещать диалоговое окно в пределах окна браузера. По умолчанию значение этой опции равно `true`, и я рекомендую не менять эту настройку. Благодаря этому пользователь при необходимости всегда сможет увидеть основную часть содержимого, заслоненную диалоговым окном. Это особенно важно, когда диалоговое окно используется для вывода сообщения о возникновении какой-либо ошибки или проблемы. Если опция `draggable` равна `false`, то пользователь не сможет сместить диалоговое окно в сторону.

Кроме того, опция `draggable` может быть полезной в ситуациях, когда вы используете несколько диалоговых окон в одном и том же окне браузера. Я рекомендую всячески этого избегать, но если такая необходимость все-таки возникает,

следует принимать меры к тому, чтобы пользователь смог изменять расположение окон, делая доступным для чтения содержимое каждого из них. На небольшом экране диалоговые окна перекрывают друг друга. Другой полезной опцией является опция `stack`, которая позволяет задать перемещение диалогового окна на передний план при выполнении на нем щелчка. По умолчанию значением этой опции также является `true`. Пример использования обеих опций приведен в листинге 22.6.

Листинг 22.6. Использование опций `draggable` и `stack`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#dialog').dialog({
                stack: true,
                draggable: true
            });
            $('#d1').dialog("option", "draggable", false);
            $('#d2').dialog("option", "stack", false);
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="d1" class="dialog" title="Первое диалоговое окно">
      Это первое диалоговое окно
    </div>
    <div id="d2" class="dialog" title="Второе диалоговое окно">
      Это второе диалоговое окно
    </div>
    <div id="d3" class="dialog" title="Третье диалоговое окно">
      Это третье диалоговое окно
    </div>
</body>
</html>
```

В этом документе создаются три диалоговых окна. Для одного из них мы отключаем опцию `draggable`, а для другого — опцию `stack`. Результат представлен на рис. 22.6, но по-настоящему понять, насколько неудачны внесенные нами изменения, вы сможете только в том случае, если поработаете с документом, открыв его в окне браузера.

Создание модальных диалоговых окон

Модальное диалоговое окно лишает пользователя возможности взаимодействовать с документом до тех пор, пока оно не будет закрыто. Для создания модального диалогового окна следует использовать опцию `modal` со значением `true`, как показано в листинге 22.7.

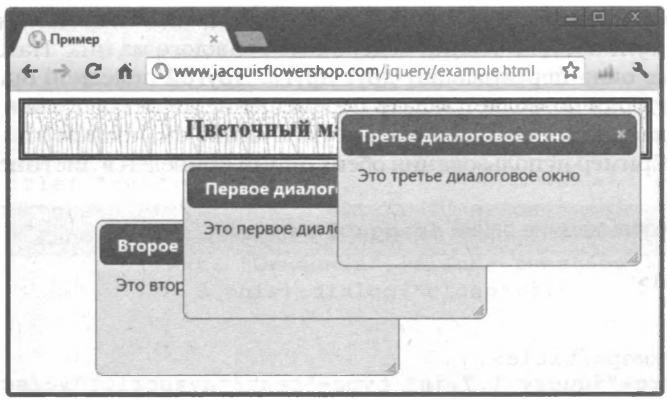


Рис. 22.6. Последствия отключения опций `draggable` и `stack`

Листинг 22.7. Создание модального диалогового окна

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#dialog').dialog({
                buttons: [{text: "OK", click: function() {
                    $(this).dialog("close")}}],
                modal: true,
                autoOpen: false
            })

            $('#show').button().click(function() {
                $('#dialog').dialog("open");
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dialog" title="Модальное диалоговое окно">
      Это модальное диалоговое окно. Для продолжения
      нажмите кнопку OK.
    </div>
</body>
</html>
```

В этом примере создается диалоговое окно, которое первоначально невидимо для пользователя. Диалоговое окно отображается в ответ на выполнение пользователем щелчка на кнопке. Результат представлен на рис. 22.7. В основе этого примера лежат

методы open и close, которые используются для отображения и сокрытия диалогового окна. О методах, поддерживаемых виджетом Dialog, будет говориться далее.

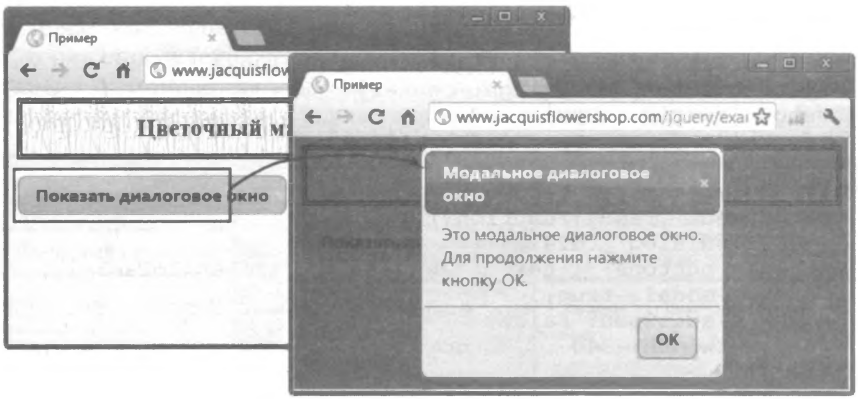


Рис. 22.7. Отображение модального диалогового окна

При отображении модального диалогового окна библиотека jQuery UI помещает полупрозрачный темный слой между диалоговым окном и остальной частью документа. Документ не вернется в свое исходное состояние до тех пор, пока не закроется диалоговое окно. В данном примере пользователь может сделать это, щелкнув на кнопке OK.

Совет. Если в само диалоговое окно также добавляются кнопки, то необходимо следить за тем, чтобы для выбора кнопки, которая была добавлена в документ для отображения диалогового окна, не использовался селектор `$('button')`. Этому селектору соответствуют все кнопки — как добавленные вами, так и созданные при вызове метода `dialog()`. Это означает, что кнопки диалогового окна будут связаны с тем же обработчиком события `click`, что и кнопка в документе, а не с функциями-обработчиками, указанными в опции `buttons`.

Отображение формы в модальном диалоговом окне

Диалоговые окна имеют то преимущество, что они привлекают внимание пользователей. Этим можно воспользоваться для отображения форм в модальных окнах, как показано в листинге 22.8.

Листинг 22.8. Отображение формы в модальном диалоговом окне

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
```

```
    <style type="text/css">
        #dialog input {width: 150px; margin: 5px;
            text-align: left}
        #dialog label {width: 100px}
        table {border-collapse: collapse;
            border: thin solid black; margin: 10px}
        #placeholder {text-align: center}
        #show {margin: 10px}
        td, th {padding: 5px; width: 100px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#dialog').dialog({
                buttons: [{text: "OK", click: addDataToTable}],
                modal: true,
                autoOpen: false,
                width: 340
            })

            $('#show').button().click(function() {
                $('#dialog').dialog("open");
            })

            function addDataToTable() {
                var data = {
                    product: $('#product').val(),
                    color: $('#color').val(),
                    count: $('#count').val()
                }
                $('#placeholder').hide();
                $('#rowTmpl').tmpl(data).appendTo('#prods tbody');
                $('#dialog').dialog("close");
            }
        });
    </script>
    <script id="rowTmpl" type="text/x-jquery-tmpl">
        <tr><td>${product}</td><td>${color}</td>
        <td>${count}</td></tr>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dialog" title="Введите данные" class="ui-widget">
        <div><label for="product">Продукт: </label>
            <input id="product" /></div>
        <div><label for="color">Цвет: </label>
            <input id="color" /></div>
        <div><label for="count">Количество: </label>
            <input id="count" /></div>
    </div>
    <table id="prods" class="ui-widget" border="1">
        <tr><th>Продукт</th><th>Цвет</th><th>Количество</th></tr>
        <tr id="placeholder"><td colspan=3>
            Не выбран ни один продукт</td></tr>
    </table>
    <button id="show">Добавить продукт</button>
</body>
</html>
```

В этом примере внутри элемента `div`, на основе которого создается диалоговое окно, определен простой набор элементов `input`. После щелчка на кнопке, расположенной в документе, отображается диалоговое окно, с помощью которого пользователь может ввести необходимые данные. Когда пользователь щелкнет на кнопке OK (которая определена с помощью опции `buttons`), введенные им данные будут обработаны с помощью шаблона данных для генерации новой строки в HTML-таблице. Соответствующая последовательность событий отражена на рис. 22.8.

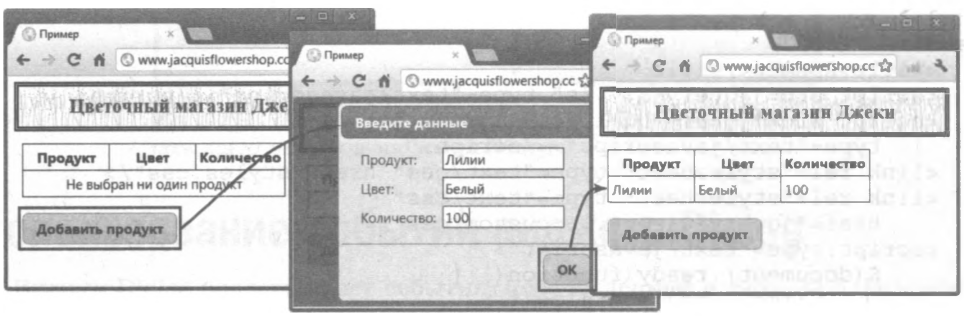


Рис. 22.8. Использование модального диалогового окна для получения данных от пользователя

Я не хотел усложнять этот пример, но в реальных проектах можно применить методики проверки корректности введенных пользователем данных, описанные в главе 13, а также методики Ajax, описанные в главах 14 и 15, для отправки данных на удаленный сервер.

Отображение формы в модальном диалоговом окне удобно использовать лишь в случае простых форм. Пытаясь сочетать с диалоговым окном вкладки Tabs или панели Accordion, вы рискуете запутать пользователя и вызвать его недовольство. Если заполнение формы требует заметных усилий, то следует подумать о том, чтобы интегрировать ее непосредственно в документ.

Использование методов виджета Dialog

Методы, поддерживаемые виджетом jQuery UI Dialog, перечислены в табл. 22.4.

Таблица 22.4. Методы виджета Dialog

Метод	Описание
`dialog("destroy")`	Удаляет виджет Dialog из базового элемента
`dialog("disable")`	Отключает диалоговое окно
`dialog("enable")`	Включает диалоговое окно
`dialog("option")`	Изменяет одну или несколько опций
`dialog("close")`	Закрывает диалоговое окно
`dialog("isOpen")`	Возвращает `true`, если диалоговое окно видимо на экране
`dialog("moveToTop")`	Перемещает диалоговое окно на вершину стека
`dialog("open")`	Отображает диалоговое окно для пользователя

Как можно было догадаться, большинство этих методов обеспечивают управление диалоговым окном из программы. Разумеется, чаще всего используются методы open и close. Пример использования наиболее важных методов приведен в листинге 22.9.

Листинг 22.9. Использование методов виджета Dialog

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#d1, #d2').dialog({
                autoOpen: false,
                position: ["right", "top"]
            }).filter("#d2")
              .dialog("option", "position", ["right", "bottom"])

            $('#t1, #t2').button().click(function (e) {
                var target = this.id == "t1" ? "#d1" : "#d2";
                if ($(target).dialog("isOpen")) {
                    $(target).dialog("close")
                } else {
                    $(target).dialog("open")
                }
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="d1" class="dialog" title="Первое диалоговое окно"
        class="ui-widget">Это первое диалоговое окно
    </div>
    <div id="d2" class="dialog" title="Второе диалоговое окно"
        class="ui-widget">Это второе диалоговое окно
    </div>
    <div>
        <button id="t1">Переключить диалоговое окно 1</button>
    </div>
    <button id="t2">Переключить диалоговое окно 2</button>
</body>
</html>
```

В этом документе имеются две кнопки, позволяющие включать или отключать видимость обоих диалоговых окон. Состояние видимости каждого окна определяется с помощью метода isOpen(). Документ с двумя диалоговыми окнами, отображаемыми в окне браузера, показан на рис. 22.9.

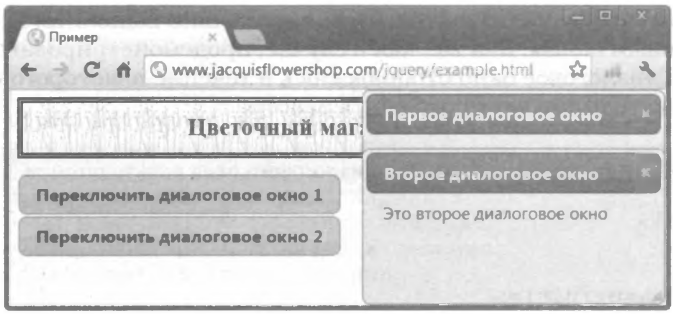


Рис. 22.9. Переключение видимости диалоговых окон с помощью методов виджета Dialog

Использование событий виджета Dialog

Виджет Dialog поддерживает события, перечисленные в табл. 22.5. Некоторые наиболее полезные события описаны в следующих разделах.

Таблица 22.5. События виджета Dialog

Событие	*Описание*
`create`	Происходит, когда виджет Dialog применяется к базовому HTML-элементу
`beforeClose`	Происходит непосредственно перед закрытием диалогового окна. Возврат значения `false` функцией — обработчиком события принудительно оставляет диалоговое окно открытым
`open`	Происходит при открытии диалогового окна
`focus`	Происходит при получении фокуса диалоговым окном
`dragStart`	Происходит, когда пользователь начинает перетаскивать диалоговое окно
`drag`	Происходит при каждом перемещении мыши в процессе перетаскивания диалогового окна
`dragStop`	Происходит по окончании перетаскивания пользователем диалогового окна
`resizeStart`	Происходит, когда пользователь начинает изменять размер диалогового окна
`resize`	Происходит при каждом перемещении мыши в процессе изменения размера диалогового окна
`resizeStop`	Происходит по окончании изменения пользователем размеров диалогового окна
`close`	Происходит при закрытии диалогового окна

Поддержание диалогового окна в открытом состоянии

Событие `beforeClose` позволяет получить уведомление о том, что пользователь затребовал закрытие диалогового окна. Это может быть вызвано тем, что пользователь нажал клавишу <Esc> (если опция `closeOnEscape` установлена равной `true`), щелкнул на кнопке закрытия, находящейся в правом верхнем углу диалогового окна, или щелкнул на кнопке, добавленной с помощью опции `buttons`.

В большинстве случаев вы будете идти навстречу пожеланиям пользователей, не препятствуя закрытию диалогового окна. Однако иногда может возникать необ-

ходимость в том, чтобы пользователь предварительно выполнил некоторые действия с диалоговым окном, или же, как в случае, продемонстрированном в листинге 22.10, чтобы диалоговое окно отображалось в течение некоторого времени, прежде чем пользователь сможет продолжить работу.

Листинг 22.10. Предотвращение закрытия диалогового окна пользователем

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 150px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            var canClose = false;
            var delay = 15;

            $('#dialog').dialog({
                modal: true,
                autoOpen: false,
                beforeClose: function() {
                    return canClose;
                },
                open: function() {
                    var count = delay;
                    var intID = setInterval(function() {
                        count--;
                        $('#time').text(count);
                        if (count == 0) {
                            clearInterval(intID)
                            canClose = true;
                            $('#dialog').dialog("close")
                        }
                    }, 1000)
                }
            })

            $('button').click(function(e) {
                $('#dialog').dialog("open")
            })

        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
```

```
    <div class="ui-widget">
        <label for="user">Имя пользователя: </label>
            <input id="user"/>
        <label for="pass">Пароль: </label>
            <input id="pass"/>
        <button id="send">Войти</button>
    </div>
    <div id="dialog" title="Неверный пароль">
        Введенный вами пароль оказался неверным.
        Пожалуйста, повторите попытку в течение
        <span id="time">15</span> секунд.
    </div>
</body>
</html>
```

В этом примере определены два элемента `input`, которые служат для получения имени пользователя и пароля. Однако в данном случае не важно, что введено в этих полях, поскольку после щелчка на кнопке Войти всегда отображается модальное диалоговое окно `Неверный пароль`.

При наступлении события `open` запускается периодически возобновляемый вызов функции, выполняющей обратный отсчет за 15 секунд. Используя событие `beforeClose`, мы лишаем пользователя возможности закрыть диалоговое окно в течение этого времени. Спустя 15 секунд вызывается метод `close`, и диалоговое окно автоматически закрывается. Путем совместного использования событий `open` и `beforeClose` мы добиваемся того, что пользователь не сможет сразу же повторить попытку ввода другого имени пользователя или пароля (по крайней мере, без повторной загрузки HTML-документа).

Реагирование на изменение размеров и положения диалогового окна

Виджет Dialog предоставляет исчерпывающий набор событий для отслеживания изменения размеров или перетаскивания диалогового окна. Обычно необходимости в использовании этих событий не возникает, но они очень полезны в тех редких ситуациях, когда важно отслеживать эти изменения. Пример использования событий `dragStart` и `dragStop` для отключения элементов `input` и `button` в документе на промежуток времени, в течение которого перетаскивается диалоговое окно, приведен в листинге 22.11.

Листинг 22.11. Реагирование на перетаскивание диалогового окна

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        input {width: 150px; text-align: left}
    </style>
```

```
    <script type="text/javascript">
        $(document).ready(function() {

            $('#dialog').dialog({
                autoOpen: true,
                dragStart: function() {
                    $('input, #send').attr("disabled", "disabled")
                },
                dragStop: function() {
                    $('input, #send').removeAttr("disabled")
                }
            })

            $('button').click(function(e) {
                $('#dialog').dialog("open")
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div class="ui-widget">
        <label for="user">Имя пользователя: </label>
            <input id="user"/>
        <label for="pass">Пароль: </label>
            <input id="pass"/>
        <button id="send">Войти</button>
    </div>
    <div id="dialog" title="Неверный пароль">
        Введенный вами пароль оказался неверным.
        Пожалуйста, повторите попытку в течение
        <span id="time">15</span> секунд.
    </div>
</body>
</html>
```

Резюме

В этой главе вы познакомились с виджетом jQuery UI Dialog. Следуя то подходу, который использовался при рассмотрении других виджетов, я сфок вал ваше внимание на опциях, методах и событиях, поддерживаемых вид Dialog. Было показано, как создавать модальные и немодальные диалоговые как отображать и скрывать диалоговые окна в ответ на события, связанные гими элементами, и как предотвратить закрытие диалогового окна пользова в течение фиксированного времени. Кроме того, было продемонстрирова пользование диалоговых окон для представления форм пользователям — мет которая при умелом применении делает работу с формами значительно удобной для пользователей.

ГЛАВА 23

Использование взаимодействий, связанных с перетаскиванием

Помимо виджетов, с которыми вы познакомились в главах 18–22, библиотека jQuery UI включает набор так называемых *взаимодействий* (interactions). Это низкоуровневые строительные блоки, позволяющие добавлять функциональность в интерфейс веб-приложений. В этой главе описываются взаимодействия Draggable и Droppable, обеспечивающие возможность перемещения элементов с помощью указателя мыши и их вставки в другие места в HTML-документе.

Взаимодействия характеризуются той же базовой структурой, что и виджеты. Они имеют свойства, методы и события. При их рассмотрении я буду следовать прежней схеме изложения с тем исключением, что иногда мне придется отклоняться от нее для обсуждения специфических особенностей некоторых взаимодействий.

Продемонстрировать с помощью экранных снимков эффекты, создаваемые взаимодействиями, довольно затруднительно. Как следует из самого их названия, они оказывают непосредственное воздействие на поведение элементов. Я старался передавать суть того, что при этом происходит, но по-настоящему понять, что собой представляют взаимодействия, вы сможете только в том случае, если самостоятельно проверите их работу в браузере. Все примеры, приведенные в данной главе (равно как и примеры в других главах), доступны для скачивания на сайте книги (см. главу 1).

Перечень тем, рассматриваемых в данной главе, приведен в табл. 23.1.

Таблица 23.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Применение взаимодействия Draggable	Используйте метод `draggable()`	1
Ограничение возможных направлений перемещения элемента	Используйте опцию `axis`	2
Ограничение области, в пределах которой можно перемещать элемент	Используйте опцию `containment`	3
Ограничение области перемещения ячейками сетки	Используйте опцию `grid`	4

Окончание табл. 23.1

Задача	Решение	Листинг
Задержка перемещения на некоторое время и наложение ограничений на расстояние перетаскивания	Используйте опции `delay` и `distance`	5
Реагирование на перемещение элемента с помощью указателя мыши	Используйте события `start`, `drag` и `stop`	6
Применение взаимодействия Droppable	Используйте метод `droppable()`	7
Подсвечивание принимающего элемента при перетаскивании перемещаемого элемента	Используйте события `activate` и `deactivate`	8
Реагирование на перекрывание перемещаемого и принимающего элементов	Используйте события `over` и `out`	9
Определение того, какие перемещаемые элементы разрешено получать принимающему элементу	Используйте опцию `accept`	10
Автоматическое применение классов CSS к принимающему элементу с началом перетаскивания перемещаемого элемента или при перекрывании перемещаемого и принимающего элементов	Используйте опции `activeClass` и `hoverClass`	11
Изменение степени перекрывания, при которой происходит событие `over`	Используйте опцию `tolerance`	12
Создание групп совместимых перемещаемых и принимающих элементов	Используйте опцию `scope`	13
Вставка перемещаемого элемента в нужное место документа во время или после перемещения	Используйте опцию `helper`	14, 15
Манипулирование вспомогательным элементом в ответ на события принимающего элемента	Используйте свойство `ui.helper`	16
Принудительная привязка перемещаемого элемента к краям других элементов	Используйте опции `snap`, `snapMode` и `snapTolerance`	17

Создание взаимодействия Draggable

Элемент, к которому применено взаимодействие Draggable, становится *перемещаемым* (draggable), т.е. его можно перемещать с помощью указателя мыши в пределах окна браузера. При открытии документа такой элемент появляется в макете на своем обычном месте, но после этого пользователь сможет изменить его местоположение путем щелчка на нем мышью и перетаскивания его в другое место. Простой пример этого вида взаимодействия приведен в листинге 23.1.

Листинг 23.1. Использование взаимодействия Draggable

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
```

```
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #draggable {font-size: x-large; border: thin solid black;
            width: 5em; text-align: center}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#draggable').draggable();
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="draggable">
        Перетащи меня
    </div>
</body>
</html>
```

В этом примере мы выбираем элемент `div` и делаем его перемещаемым путем вызова для него метода `draggable()`. Как показано на рис. 23.1, в открывшемся документе элемент занимает свою обычную позицию, но после этого его можно переместить с помощью указателя мыши в любое место в окне браузера. Обратите внимание на то, что на другие элементы документа, например на элемент `h1`, вызов метода `draggable()` никак не влияет.

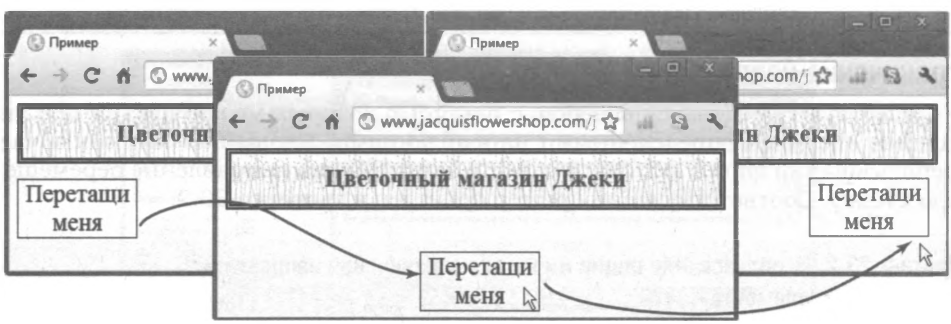


Рис. 23.1. Перемещение элемента в окне браузера с помощью указателя мыши

Совет. Возможность перетаскивания элементов полезна уже сама по себе, но она приносит еще больше пользы, если применяется в сочетании с взаимодействием Droppable, которое описано далее.

Взаимодействие Draggable реализуется исключительно за счет использования специфической HTML-разметки и CSS-стилей. Это означает, что данная функциональность будет работать практически в любом браузере, но наделенные ею элементы не смогут работать с аналогичными собственными средствами *Drag-and-drop* операционных систем.

Совет. Определяемые спецификацией HTML5 операции *Drag-and-drop* обычно реализуются с использованием собственных механизмов операционных систем. Если вы используете механизм *Drag-and-drop* jQuery UI, то во избежание возникновения конфликтных ситуаций эквивалентные средства HTML5 лучше отключить. С этой целью установите для атрибута `draggable` элемента `body` документа значение `false`.

Настройка взаимодействия Draggable

Существует множество опций настройки для взаимодействия Draggable. Наиболее важные свойства, рассмотрению которых посвящены следующие разделы, приведены в табл. 23.2.

Таблица 23.2. Свойства взаимодействия Draggable

Свойство	*Описание*
`axis`	Ограничивает возможности перемещения определенными направлениями. Значение по умолчанию — `false`; оно означает отсутствие ограничений, но можно также указать значение `x` (перемещение только вдоль оси `x`) или `y` (перемещение только вдоль оси `y`)
`containment`	Ограничивает местоположение перемещаемого элемента определенной областью экрана. Типы поддерживаемых значений описаны в табл. 23.3. Значение по умолчанию — `false`; оно означает отсутствие ограничений
`delay`	Определяет время, в течение которого должно осуществляться перетаскивание элемента, прежде чем он переместится. Значение по умолчанию — `0`; оно означает отсутствие задержки
`distance`	Определяет расстояние, на которое пользователь должен перетащить элемент из его начальной позиции, прежде чем он действительно переместится. Значение по умолчанию — `1` пиксель
`grid`	Осуществляет принудительную привязку перемещаемого элемента к ячейкам сетки. Значение по умолчанию — `false`; оно означает отсутствие привязки

Ограничение направлений перемещения

Существуют несколько способов, с помощью которых можно ограничить перемещение элемента определенными направлениями. Первый из них заключается в использовании опции `axis`, позволяющей ограничить направление перемещения осью `x` или `y`. Соответствующий пример приведен в листинге 23.2.

Листинг 23.2. Использование опции `axis` для ограничения направления перемещения элемента

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.dragElement {font-size: large;
            border: thin solid black; width: 6.5em;
            text-align: center; background-color: lightgray;
            margin: 4px }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
```

```
            $('.dragElement').draggable({
                axis: "x"
            }).filter('#dragV').draggable("option", "axis", "y");

        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dragV" class="dragElement">
        Перетащить по вертикали
    </div>
    <div id="dragH" class="dragElement">
        Перетащить по горизонтали
    </div>
</body>
</html>
```

В этом примере мы определяем два элемента `div`, выбираем их с помощью jQuery и вызываем метод `draggable()`. В качестве аргумента этому методу передается объект, который первоначально ограничивает перемещение обоих элементов `div` направлением вдоль оси `x`. Применив затем метод jQuery `filter()`, мы получаем возможность выбрать элемент `dragV` без повторного поиска средствами jQuery по всему документу и установить для него другое разрешенное направление перемещения — вдоль оси `y`. Таким образом, мы получаем документ, в котором один элемент `div` можно перетаскивать только в вертикальном направлении, а другой — только в горизонтальном. Результат представлен на рис. 23.2.

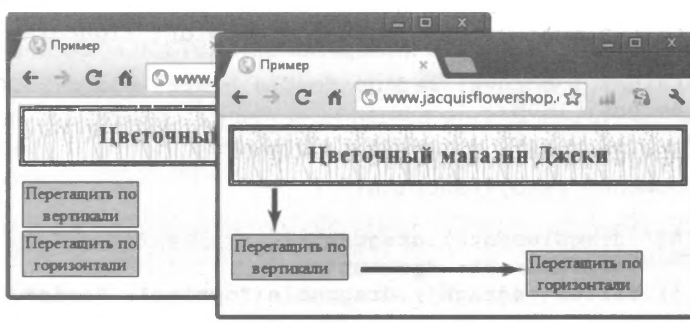


Рис. 23.2. Ограничение направлений, в которых можно перетаскивать элементы

Ограничение допустимой области перемещения элемента

Можно также ограничить область экрана, в которой допускается перетаскивание элемента. Для этого используется опция `containment`. Форматы значений, которые можно указывать в этой опции, описаны в табл. 23.3.

Таблица 23.3. Типы значений опции `containment`

Значение	*Описание*
Селектор	Если указан селектор, перетаскивание перемещаемого элемента ограничивается областью, занимаемой первым подходящим элементом

Окончание табл. 23.3

Значение	Описание
HTML-элемент	Перетаскивание ограничивается областью, занимаемой указанным элементом
Строка	Для ограничения области перетаскивания можно указать строку, содержащую одно из следующих значений: `parent`, `document`, `window`
Числовой массив	Область перетаскивания можно ограничить, определив ее числовым массивом формата `[x1, y1, x2, y2]`

Пример использования опции `containment` приведен в листинге 23.3.

Листинг 23.3. Использование опции `containment`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.dragElement {font-size: large;
            border: thin solid black; width: 6.5em;
            text-align: center; background-color: lightgray;
            margin: 4px }
        #container { border: medium double black; width: 400px;
            height: 150px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('.dragElement').draggable({
                containment: "parent"
            }).filter('#dragH').draggable("option", "axis", "x");

        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="container">
        <div id="dragH" class="dragElement">
            Перетащить по горизонтали
        </div>
        <div class="dragElement">
            Перетащить внутри родителя
        </div>
    </div>
</body>
</html>
```

В этом примере возможности перемещения обоих элементов ограничены таким образом, что их можно перетаскивать только внутри родительского элемента, в качестве которого выступает элемент div с фиксированными размерами. Для одного из перемещаемых элементов div с помощью опции axis введено дополнительное ограничение, заключающееся в том, что он может перемещаться внутри родительского элемента только в горизонтальном направлении. Результат проиллюстрирован на рис. 23.3.

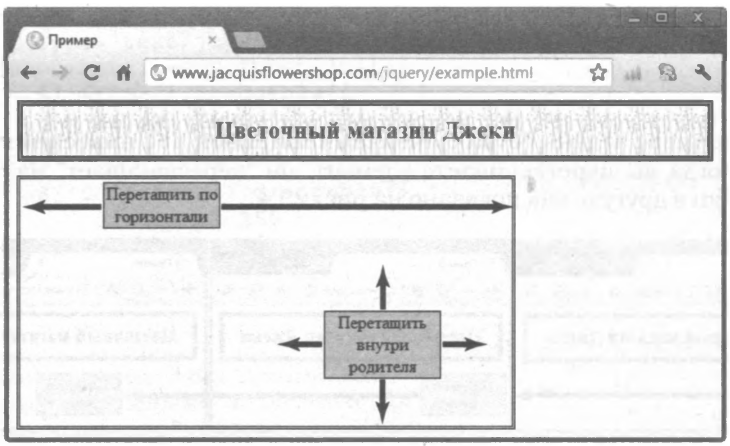


Рис. 23.3. Ограничение области перетаскивания пространством родительского элемента

Ограничение возможностей перемещения элемента ячейками сетки

Опция grid позволяет задать привязку перемещаемого элемента к ячейкам сетки. Эта опция принимает в качестве значения массив из двух элементов, определяющих ширину и высоту ячеек сетки в пикселях. Пример использования опции grid приведен в листинге 23.4.

Листинг 23.4. Использование опции grid

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #draggable {font-size: large; border: thin solid black;
            padding: 4px; width: 100px; text-align: center;
            background-color: lightgray; margin: 4px; }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#draggable').draggable({
```

```
                grid: [100, 50]
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="draggable">
        Перетащи меня
    </div>
</body>
</html>
```

В этом примере задана сетка с ячейками шириной `100` пикселей и высотой `50` пикселей. Когда вы перетаскиваете элемент, он "перескакивает" из одной (невидимой) ячейки в другую, как показано на рис. 23.4.

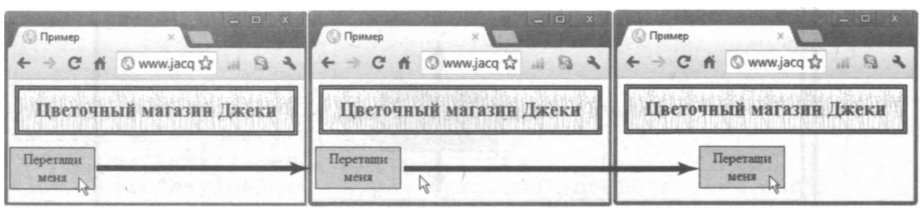


Рис. 23.4. Перетаскивание элемента по ячейкам сетки

Эффект привязки является весьма показательным примером использования функциональности взаимодействий, однако его трудно передать с помощью экранных снимков.

Совет. Можно создать эффект привязки лишь для одного направления, указав для оси свободного перемещения значение `1`. Например, если присвоить опции `grid` значение `[100, 1]`, то элемент будет привязываться к ячейкам сетки шириной `100` пикселей при перемещении по горизонтали, но перемещении по вертикали будет свободным.

Задержка перемещения

Существуют две опции, позволяющие организовать задержку при перетаскивании перемещаемого элемента. С помощью опции `delay` можно задать время в миллисекундах, в течение которого пользователь должен перетаскивать указатель мыши, прежде чем элемент будет действительно перемещен. Другой вид задержки обеспечивается опцией `distance`, определяющей расстояние в пикселях, на которое пользователь должен перетащить указатель мыши, прежде чем за ним последует элемент. Пример использования обеих настроек приведен в листинге 23.5.

Листинг 23.5. Использование опций `delay` и `distance`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
```

```
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #time, #distance {font-size: large;
            border: thin solid black; padding: 4px;
            width: 100px; text-align: center;
            background-color: lightgray; margin: 4px; }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#time').draggable({
                delay: 1000
            })

            $('#distance').draggable({
                distance: 150
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="time">Время задержки</div>
    <div id="distance">Расстояние</div>
</body>
</html>
```

В этом примере есть два перемещаемых элемента, для одного из которых задержка задана с помощью опции `delay`, а для другого — с помощью опции `distance`.

В случае задержки, определяемой опцией `delay`, пользователь должен выполнять перетаскивание в течение заданного времени, прежде чем это приведет к действительному перемещению элемента. В данном примере длительность этого промежутка составляет 1000 мс. Перемещать мышь в это время вовсе не обязательно, но на протяжении всего периода задержки кнопка мыши должна оставаться в нажатом состоянии, после чего элемент можно будет переместить, сдвинув мышь. По истечении времени задержки перемещаемый элемент привяжется к местоположению указателя мыши с учетом ограничений, налагаемых опциями `grid`, `region` и `axis`, о которых ранее говорилось.

Опция `distance` оказывает похожее воздействие, но в этом случае пользователь должен перетащить указатель мыши не менее чем на заданное количество пикселей в любом направлении от начального местоположения элемента. После этого перемещаемый элемент скачкообразно переместится к текущему местоположению указателя.

Совет. Если применить обе настройки к одному и тому же элементу, то перемещаемый элемент не сдвинется с места до тех пор, пока не будут выполнены оба критерия задержки, т.е. пока попытка перетаскивания элемента не будет длиться в течение заданного времени *и* пока указатель мыши не переместится на заданное количество пикселей.

Использование методов взаимодействия Draggable

Все методы, определенные для взаимодействия Draggable, входят в набор базовых методов, с которыми вы уже познакомились при рассмотрении виджетов. Методы, специфические для взаимодействия Draggable, не предусмотрены. Перечень доступных методов приведен в табл. 23.4.

Таблица 23.4. Методы взаимодействия Draggable

Метод	Описание
`draggable("destroy")`	Полностью удаляет функциональность взаимодействия Draggable из элемента
`draggable("disable")`	Временно отключает функциональность взаимодействия Draggable для базового элемента
`draggable("enable")`	Включает ранее отключенную функциональность взаимодействия Draggable для базового элемента
`draggable("option")`	Позволяет получить или изменить значение одной или нескольких опций

Использование событий взаимодействия Draggable

Взаимодействие Draggable поддерживает простой набор событий, уведомляющих о перетаскивании элемента. Эти события описаны в табл. 23.5.

Таблица 23.5. События взаимодействия Draggable

Событие	Описание
`create`	Происходит в момент применения взаимодействия Draggable к элементу
`start`	Происходит в момент начала перетаскивания
`drag`	Происходит при каждом перемещении мыши в процессе перетаскивания элемента
`stop`	Происходит в момент отпускания кнопки мыши в процессе перетаскивания

Как и в случае событий виджетов, на эти события также можно реагировать. Пример обработки событий `start` и `stop` приведен в листинге 23.6.

Листинг 23.6. Использование событий `start` и `stop` взаимодействия Draggable

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #draggable {font-size: large; border: thin solid black;
            padding: 4px; width: 150px; text-align: center;
            background-color: lightgray; margin: 4px; }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#draggable').draggable({
                start: function() {
                    $('#draggable').text("Перетаскивание...")
                },
                stop: function() {
```

```
                $('#draggable').text("Перетащи меня")
            }
        })
    });
</script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="draggable">
        Перетащи меня
    </div>
</body>
</html>
```

В этом примере события `start` и `stop` используются для изменения текстового содержимого элемента в процессе перетаскивания. Эта благоприятная возможность является следствием того, что взаимодействие Draggable реализовано исключительно с использованием средств HTML и CSS: можно использовать jQuery для изменения состояния перемещаемого элемента даже в то время, когда он движется по экрану. Результат проиллюстрирован на рис. 23.5.

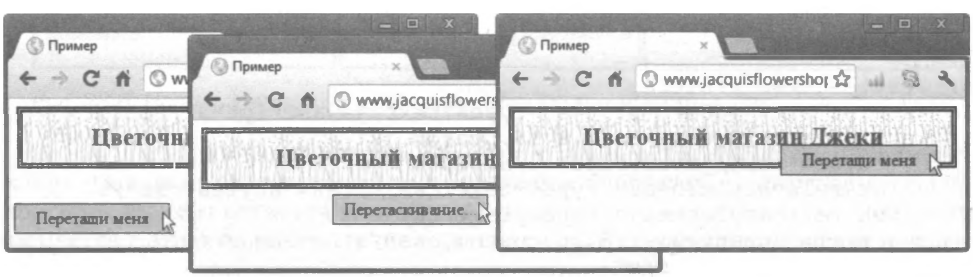


Рис. 23.5. Использование событий взаимодействия Draggable для изменения свойств элемента при его перетаскивании

Использование взаимодействия Droppable

В некоторых ситуациях одной лишь возможности перетаскивания элемента может быть вполне достаточно, но наибольшую пользу оно приносит в тех случаях, когда используется совместно с взаимодействием Droppable.

Элементы, к которым было применено взаимодействие Droppable (*принимающие* элементы), приобретают способность принимать *перемещаемые* элементы, созданные с помощью взаимодействия Draggable.

Принимающие элементы создаются с помощью метода `droppable()`, но для получения полезной функциональности потребуется создать обработчики событий из числа тех, которые определены для этого вида взаимодействия. Доступные события приведены в табл. 23.6.

Таблица 23.6. События взаимодействия Droppable

Событие	*Описание*
`create`	Происходит в момент применения взаимодействия Droppable к элементу
`activate`	Происходит, когда пользователь начинает перетаскивать перемещаемый элемент

Окончание табл. 23.6

Событие	Описание
deactivate	Происходит, когда пользователь прекращает перетаскивать перемещаемый элемент
over	Происходит, когда пользователь перетаскивает перемещаемый элемент над принимающим элементом (но при условии, что кнопка мыши еще не была отпущена)
out	Происходит, когда пользователь перетаскивает перемещаемый элемент за пределы принимающего элемента
drop	Происходит, когда пользователь оставляет перемещаемый элемент на принимающем элементе

Пример создания простого принимающего элемента, для которого определен единственный обработчик события `drop`, приведен в листинге 23.7.

Листинг 23.7. Создание простого взаимодействия Droppable

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #draggable, #droppable {font-size: large;
            border: thin solid black; padding: 10px; width: 100px;
            text-align: center; background-color: lightgray;
            margin: 4px;}
        #droppable {padding: 20px; position: absolute;
            right: 5px;}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#draggable').draggable();

            $('#droppable').droppable({
                drop: function() {
                    $('#draggable').text("Оставлен")
                }
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="droppable">
        Оставь здесь
    </div>
    <div id="draggable">
        Перетащи меня
```

```
    </div>
</body>
</html>
```

В этом примере в документ добавлен элемент `div`, текстовое содержимое которого представлено строкой `Оставь здесь`. Мы выбираем этот элемент, используя jQuery, и вызываем метод `droppable()`, передавая ему объект с настройками, который определяет обработчик для события `drop`. Ответом на это событие является изменение текста перемещаемого элемента с помощью метода `text()`. Создаваемое в данном примере интерактивное взаимодействие категории *Drag-and-drop* является простейшим, но оно создает удобный контекст для объяснения возможностей совместной работы взаимодействий Draggable и Droppable. Различные стадии процесса перетаскивания элементов проиллюстрированы на рис. 23.6.

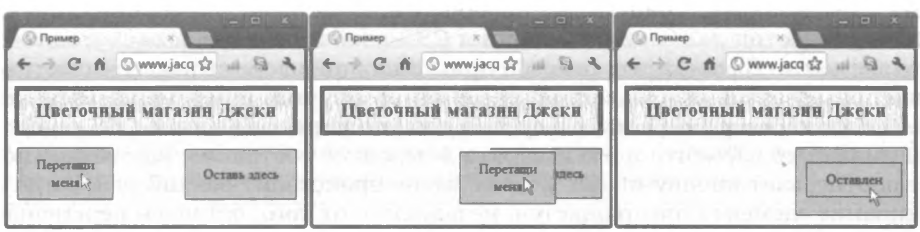


Рис. 23.6. Использование простой операции Drag-and-drop

Все это выглядит очень просто. Мы перетаскиваем перемещаемый элемент до тех пор, пока он не окажется над принимающим элементом, и отпускаем его. Перемещаемый элемент остается там, где он был оставлен, и его текстовое содержимое изменяется в ответ на наступление события `drop`. В следующих разделах показано, как использовать другие события взаимодействия Droppable для повышения комфортности работы пользователя.

Подсветка целевого принимающего объекта

Используя события `activate` и `deactivate`, можно подсветить целевой принимающий объект, когда пользователь начинает процесс перетаскивания элемента. Во многих ситуациях эта идея оказывается весьма плодотворной, поскольку при этом пользователь получает надежное указание относительно того, какие элементы являются частью модели *Drag-and-drop*. Соответствующий пример приведен в листинге 23.8.

Листинг 23.8. Реагирование на события `activate` и `deactivate`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#draggable').draggable();

        $('#droppable').droppable({
            drop: function() {
                $('#draggable').text("Оставлен");
            },
            activate: function() {
                $('#droppable').css({
```

```
                    border: "medium double black",
                    backgroundColor: "cyan"
                });
            },
            deactivate: function() {
                $('#droppable').css("border", "")
                    .css("background-color", "");
            }
        });
    });
</script>
...
```

Как только пользователь начинает перетаскивать элемент, срабатывает событие `activate`, связанное с нашим принимающим элементом, и функция-обработчик использует метод `css()` для изменения CSS-свойств `border` и `background-color` этого элемента. В результате целевой принимающий элемент подсвечивается, указывая пользователю на существование связи между ним и перемещаемым элементом. Событие `deactivate` используется для удаления значений CSS-свойств из принимающего элемента и его возврата в исходное состояние, как только пользователь отпускает кнопку мыши. (Это событие происходит всякий раз, когда перетаскивание элемента прекращается, независимо от того, оставлен перемещаемый элемент на принимающем элементе или не оставлен.) Этот процесс проиллюстрирован на рис. 23.7.

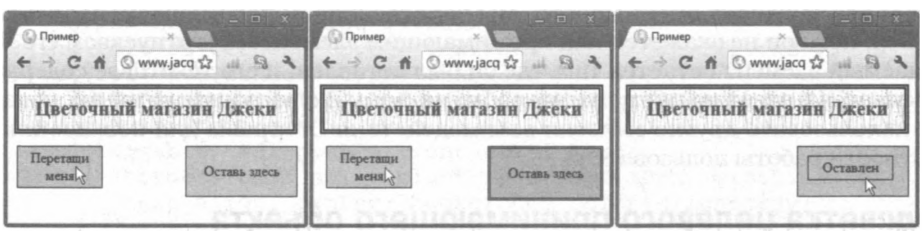


Рис. 23.7. Использование событий `activate` и `deactivate`

Обработка перекрывания элементов

Технологию *Drag-and-drop* можно усовершенствовать, добавив в нее обработку событий `over` и `out`. Событие `over` происходит, когда 50% перемещаемого элемента оказывается над любой частью принимающего элемента. Событие `out` наступает тогда, когда перекрывавшиеся ранее элементы перестают перекрываться. Пример ответной реакции на эти события приведен в листинге 23.9.

Листинг 23.9. Использование событий `over` и `out`

```
<script type="text/javascript">
    $(document).ready(function() {
        $('#draggable').draggable();

        $('#droppable').droppable({
            drop: function() {
                $('#draggable').text("Оставлен");
            },
```

```
            over: function() {
                $('#droppable').css({
                    border: "medium double black",
                    backgroundColor: "cyan"
                });
            },
            out: function() {
                $('#droppable').css("border", "")
                    .css("background-color", "");
            }
        });
    });
</script>
```

Здесь использованы те же функции-обработчики, что и в предыдущем примере, но в данном случае они связаны с событиями `over` и `out`. Когда с принимающим элементом перекрывается по крайней мере 50% перемещаемого элемента, он заключается в рамку и цвет его фона изменяется, как показано на рис. 23.8.

Совет. Указанный 50%-ный предел называется *порогом перекрывания* (tolerance), величину которого можно задавать при создании принимающего элемента, как будет показано далее.

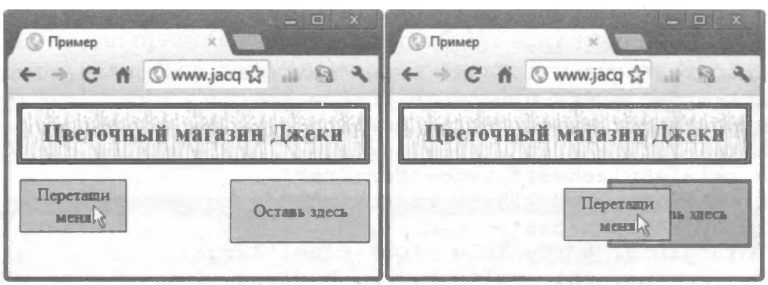


Рис. 23.8. Реагирование на события `over` и `out`

Настройка взаимодействия Droppable

В этой главе я несколько отклонился от обычной схемы изложения, к чему меня побудила важность той роли, которую играют события при работе со взаимодействием Droppable. Конечно же, для этого взаимодействия предусмотрен ряд свойств, путем изменения которых можно настроить его поведение. Эти свойства перечислены в табл. 23.7.

Таблица 23.7. Свойства взаимодействия Droppable

Свойство	*Описание*
`disabled`	Если эта опция равна `true`, то функциональность взаимодействия Droppable первоначально отключена. Значение по умолчанию — `false`
`accept`	Сужает множество перемещаемых элементов, на которые будет реагировать принимающий элемент. Значение по умолчанию — `*`; ему соответствует любой элемент
`activeClass`	Определяет класс, который будет присваиваться в ответ на событие `activate` и удаляться в ответ на событие `deactivate`

Окончание табл. 23.7

Свойство	Описание
hoverClass	Определяет класс, который будет присваиваться в ответ на событие over и удаляться в ответ на событие out
tolerance	Определяет минимальную степень перекрывания, при которой происходит событие over

Ограничение допустимых перемещаемых элементов

Можно ограничить множество перемещаемых элементов, которые будут приниматься элементом, наделенным функциональностью взаимодействия Droppable, с помощью опции accept. В качестве значения опции accept следует присвоить селектор. В результате этого события взаимодействия Droppable будут происходить лишь в том случае, если перемещаемый элемент соответствует указанному селектору. Соответствующий пример приведен в листинге 23.10.

Листинг 23.10. Ограничение множества принимаемых элементов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .draggable, #droppable {font-size: large;
            border: thin solid black; padding: 4px; width: 100px;
            text-align: center; background-color: lightgray;
            margin: 4px;}
        #droppable {padding: 20px; position: absolute;
            right: 5px;}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('.draggable').draggable();

            $('#droppable').droppable({
                drop: function(event, ui) {
                    ui.draggable.text("Оставлен");
                },
                activate: function() {
                    $('#droppable').css({
                        border: "medium double black",
                        backgroundColor: "cyan"
                    });
                },
                deactivate : function() {
                    $('#droppable').css("border", "")
                        .css("background-color", "");
                },
```

```
                accept: '#drag1'
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="droppable">
        Оставь здесь
    </div>
    <div id="drag1" class="draggable">
        Элемент 1
    </div>
    <div id="drag2" class="draggable">
        Элемент 2
    </div>
</body>
</html>
```

В этом примере есть два перемещаемых элемента с идентификаторами `drag1` и `drag2` (на рисунке это `Элемент 1` и `Элемент 2`). При создании принимающего элемента используется опция `accept`, с помощью которой мы указываем, что приемлемым перемещаемым элементом будет только элемент `drag1`. При перетаскивании элемента `drag1` вы будете наблюдать тот же эффект, что и в предыдущем примере. В соответствующие моменты для принимающего элемента будут запускаться события `activate`, `deactivate`, `over` и `out`. В то же время, если перетаскивать элемент `drag2`, который не соответствует указанному в параметре `accept` селектору, то эти события запускаться не будут. Этот элемент можно свободно перемещать, но он не будет восприниматься принимающим элементом. Данный эффект проиллюстрирован на рис. 23.9.

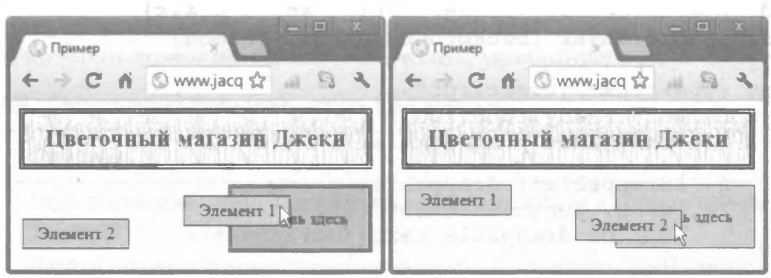


Рис. 23.9. Использование опции `accept`

Обратите внимание на изменение способа выбора приемлемого перемещаемого элемента, для которого следует вызывать метод `text()`. Когда в документе был всего лишь один перемещаемый элемент, для этого хватало атрибута `id`.

```
$('#draggable').text("Оставлен");
```

В данном примере имеется два перемещаемых элемента, и выбор по атрибуту `id` не даст желаемого результата, поскольку текст в этом случае будет всегда изменяться в одном и том же перемещаемом элементе, независимо от того, какой из них является приемлемым для принимающего элемента. Выход состоит в том, чтобы использовать объект `ui`, который jQuery UI предоставляет в качестве дополнительного аргумента каждому обработчику событий. Свойство `draggable` объекта `ui` возвращает объект `jQuery`, содержащий элемент, который пользователь перетас-

кивает или пытается оставить на целевом элементе, что позволяет выбрать требуемый элемент следующим образом:

```
ui.draggable.text("Оставлен");
```

Подсветка принимающего элемента с использованием классов

Опции activeClass и hoverClass позволяют изменить внешний вид принимающего элемента, не прибегая к событиям activate, deactivate, over и out. Соответствующий пример приведен в листинге 23.11.

Листинг 23.11. Использование опций activeClass и hoverClass

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .draggable, #droppable {font-size: large;
            border: thin solid black; padding: 4px; width: 100px;
            text-align: center; background-color: lightgray;
            margin: 4px;}
        #droppable {padding: 20px; position: absolute;
            right: 5px;}
        #droppable.active {border: thick solid black}
        #droppable.hover {background-color: salmon}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('.draggable').draggable();

            $('#droppable').droppable({
                drop: function(event, ui) {
                    ui.draggable.text("Оставлен");
                },
                activeClass: "active",
                hoverClass: "hover"
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="droppable">
        Оставь здесь
    </div>
    <div class="draggable">
        Перетащи меня
    </div>
</body>
</html>
```

В этом примере определены два новых CSS-стиля, которые выделены в листинге полужирным шрифтом. Создав классы, мы получили возможность различать элементы с одинаковыми атрибутами `id` (например, `#droppable.active`), так что новые стили более специфичны по сравнению с другими стилями (например, `#droppable`) и потому имеют более высокий приоритет. Правила применения стилей CSS к элементам подробно обсуждаются в главе 3.

Определив новые стили, мы используем имена соответствующих классов в качестве значений опций `activeClass` и `hoverClass`. Всю работу по добавлению и удалению этих классов из принимающего элемента в ответ на наступление соответствующих событий выполняет взаимодействие Droppable. Результат представлен на рис. 23.10.

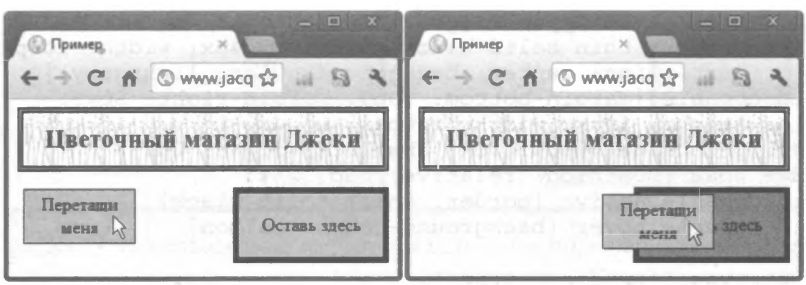


Рис. 23.10. Использование параметров `activeClass` и `hoverClass`

Изменение порога перекрывания

По умолчанию событие `over` происходит лишь в тех случаях, когда по крайней мере 50% перемещаемого элемента перекрывается с принимающим элементом. Величину этого порогового перекрывания можно изменить с помощью опции `tolerance`, которая может принимать значения, указанные в табл. 23.8.

Таблица 23.8. Допустимые значения опции `tolerance`

Значение	*Описание*
`fit`	Перетаскиваемый элемент должен полностью находиться в области принимающего элемента
`intersect`	Перетаскиваемый элемент должен перекрываться с принимающим элементом по крайней мере наполовину. Это значение используется по умолчанию
`pointer`	Указатель мыши должен находиться в области принимающего элемента, независимо от того, где именно перетаскиваемый элемент был захвачен пользователем
`touch`	Означает любую степень перекрывания перетаскиваемого и принимающего элементов

Чаще всего я использую два значения, `fit` и `touch`, поскольку их смысл наиболее понятен для пользователей. Значение `fit` используется мною в тех случаях, когда перетаскиваемый элемент должен остаться в той области принимающего элемента, в которую он был перемещен, а значение `touch` — когда перемещенный элемент должен вернуться в исходную позицию (соответствующий пример будет приведен далее). Пример использования параметров `fit` и `touch` приведен в листинге 23.12.

Листинг 23.12. Изменение порога перекрывания для перемещаемых элементов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .draggable, .droppable {font-size: large;
            border: thin solid black; padding: 4px; width: 100px;
            text-align: center; background-color: lightgray;}
        .droppable {margin-bottom: 10px; margin-right: 5px;
            height: 50px; width: 120px}
        #dropContainer {position: absolute; right: 5px;}
        div span {position: relative; top: 25%}
        .droppable.active {border: thick solid black}
        .droppable.hover {background-color: salmon}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('.draggable').draggable();

            $('div.droppable').droppable({
                drop: function(event, ui) {
                    ui.draggable.text("Оставлен");
                },
                activeClass: "active",
                hoverClass: "hover",
                tolerance: "fit"
            });

            $('#touchDrop')
                .droppable("option", "tolerance", "touch");
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dropContainer">
        <div id="fitDrop" class="droppable">
            <span>Fit</span>
        </div>
        <div id="touchDrop" class="droppable">
            <span>Touch</span>
        </div>
    </div>
    <div class="draggable">
        <span>Перетащи меня</span>
    </div>
</body>
</html>
```

В этом примере есть два принимающих элемента, один из которых сконфигурирован с использованием значения `fit` для параметра `tolerance`, а второй — с использованием значения `touch`. Перемещаемый элемент только один и используется для демонстрации эффектов перетаскивания с двумя различными значениями опции `tolerance`. Снимки делались в момент наступления события `over`. Заметим, что при определении момента перекрывания граница учитывается в обоих случаях.

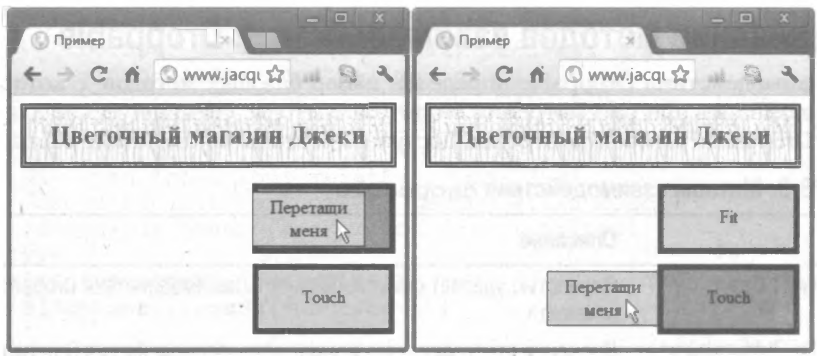


Рис. 23.11. Использование значений `fit` и `touch` параметра `tolerance`

Фиксация момента запуска события

Возможно, вас удивляет, каким образом я сумел сделать экранные снимки точно в тот момент, когда происходило событие `over`. Для этого я использовал в своем элементе `script` ключевое слово JavaScript `debugger`. Хотя в официальной спецификации языка это и не оговорено, в таких браузерах, как Google Chrome и Firefox (с установленным расширением FireBug), предусмотрены отладчики, которые приостанавливают выполнение сценария, как только в нем встречается ключевое слово `debugger` (так называемые *точки останова*). Для работы с отладчиком в сценарий добавлен обработчик события `over`, как показано ниже.

```
<script type="text/javascript">
    $(document).ready(function() {

        $('.draggable').draggable();

        $('div.droppable').droppable({
            drop: function(event, ui) {
                ui.draggable.text("Оставлен");
            },
            activeClass: "active",
            hoverClass: "hover",
            tolerance: "fit",
            over: function() {
                debugger
            }
        });

        $('#touchDrop')
            .droppable("option", "tolerance", "touch");
    });
</script>
```

Когда происходит событие over, отладчик генерирует прерывание, и дальнейшее выполнение сценария становится возможным лишь после того, как отладчик получит соответствующее указание. Это позволило мне делать экранные снимки, которые вы видите на рисунках, именно в тот момент, когда происходило событие. Как правило, ключевое слово debugger выручает во многих ситуациях, но оно становится просто незаменимым при работе с событиями. Разумеется, прежде чем предоставлять программу пользователям, все вхождения этого ключевого слова должны быть удалены из сценария.

Использование методов взаимодействия Droppable

Для взаимодействия Droppable определен набор базовых методов, с которыми вы уже познакомились при рассмотрении виджетов. Методы, специфические для взаимодействия Droppable, не предусмотрены. Доступные методы приведены в табл. 23.9.

Таблица 23.9. Методы взаимодействия droppable

Метод	*Описание*
droppable("destroy")	Полностью удаляет функциональность взаимодействия Droppable из элемента
droppable("disable")	Временно отключает функциональность взаимодействия Droppable для базового элемента
droppable("enable")	Включает ранее отключенную функциональность взаимодействия Droppable для базового элемента
droppable("option")	Позволяет изменить один или несколько параметров

Дополнительная настройка операций перетаскивания

Имеется ряд дополнительных параметров, с помощью которых можно выполнить точную настройку функциональности Draggable и Droppable jQuery UI. В данном разделе приводятся описания этих настроек и примеры их использования.

Использование опции Scope

Ранее было продемонстрировано, как использовать опцию accept для выбора перемещаемых элементов, которые могут захватываться принимающими элементами. В простых проектах селекторы работают вполне удовлетворительно, но с увеличением количества перемещаемых элементов, которыми приходится управлять, необходимые селекторы значительно усложняются, что повышает риск появления ошибок.

Альтернативный подход заключается в использовании опции scope как для перемещаемых, так и для принимающих элементов. Перемещаемый элемент будет активизировать лишь те принимающие элементы, которые имеют одинаковое с ним значение свойства scope. Пример использования этой опции приведен в листинге 23.13.

Листинг 23.13. Использование опции scope

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .draggable, .droppable {font-size: large;
            border: medium solid black; padding: 4px;
            width: 100px; text-align: center;
            background-color: lightgray; margin-bottom: 10px;}
        .droppable {margin-right: 5px; height: 50px; width: 120px}
        #dropContainer {position: absolute; right: 5px;}
        div span {position: relative; top: 25%}
        .droppable.active {border: medium solid black}
        .droppable.hover {background-color: salmon}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#apple').draggable({
                scope: "fruit"
            });
            $('#orchid').draggable({
                scope: "flower"
            });

            $('#flowerDrop').droppable({
                activeClass: "active",
                hoverClass: "hover",
                scope: "flower"
            });

            $('#fruitDrop').droppable({
                activeClass: "active",
                hoverClass: "hover",
                scope: "fruit"
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dropContainer">
        <div id="flowerDrop" class="droppable">
            <span>Цветы</span>
        </div>
        <div id="fruitDrop" class="droppable">
            <span>Фрукты</span>
        </div>
    </div>
    <div id="orchid" class="draggable">
        <span>Орхидеи</span>
    </div>
```

```
    <div id="apple" class="draggable">
        <span>Яблоки</span>
    </div>
</body>
</html>
```

В этом примере создаются два перемещаемых и два принимающих элемента. При создании им назначается одно из двух значений свойства `scope`: `fruit` или `flower`. Конечный результат состоит в том, что каждый из перемещаемых элементов будет активизировать лишь те принимающие элементы и являться приемлемым лишь для тех из них, которые имеют то же значение свойства `scope`, как показано на рис. 23.12.

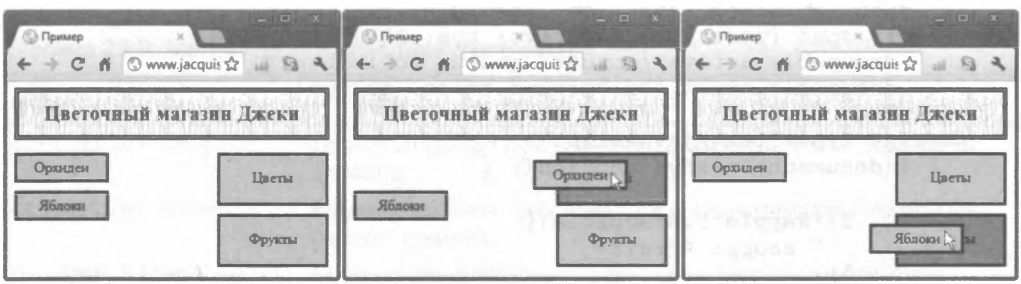


Рис. 23.12. Группировка перемещаемых и принимающих элементов по значению свойства `scope`

Совет. Обратите внимание на то, что значения свойства `scope` для каждого элемента устанавливаются при первоначальном вызове методов `draggable()` и `droppable()`, а не с использованием метода `option()`. На момент написания книги в библиотеке jQuery UI имелась ошибка, из-за которой присвоение значения свойству `scope` после создания взаимодействия не работало.

Использование вспомогательного элемента

С помощью опции `helper` можно определить вспомогательный элемент, который будет перетаскиваться вместо исходного элемента, остающегося на месте. Это разительно отличается от предыдущих примеров, в которых перемещаемый элемент покидал свою позицию. Пример использования вспомогательного элемента приведен в листинге 23.14.

Листинг 23.14. Использование перемещаемого элемента больших размеров

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
```

```
        .draggable, .droppable {font-size: large;
            border: medium solid black; padding: 4px;
            width: 150px; text-align: center;
            background-color: lightgray; margin-bottom: 10px;}
        .droppable {margin-right: 5px; height: 50px; width: 120px}
        #dropContainer {position: absolute; right: 5px;}
        div span {position: relative; top: 25%}
        .droppable.active {border: medium solid black}
        .droppable.hover {background-color: salmon}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('div.draggable').draggable({
                helper: "clone"
            });

            $('#basket').droppable({
                activeClass: "active",
                hoverClass: "hover"
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dropContainer">
        <div id="basket" class="droppable">
            <span>Корзина</span>
        </div>
    </div>
    <div class="draggable">
        <img src="lily.png"/><label for="lily">Лилии</label>
    </div>
</body>
</html>
```

Значение `clone` указывает jQuery UI на то, что необходимо создать копию перемещаемого элемента вместе со всем его содержимым и использовать полученный результат в качестве вспомогательного элемента. Результат представлен на рис. 23.13. Вспомогательный элемент удаляется, когда пользователь отпускает кнопку мыши над перемещаемым элементом, оставляя перемещаемый и принимающий элементы в их исходных позициях.

Рис. 23.13. Перемещаемый элемент с большими размерами

Как показано на рисунке, исходный перемещаемый элемент остается на своем месте и лишь вспомогательный элемент перемещается по экрану вслед за указателем мыши. Если размеры перемещаемого элемента велики, как в нашем примере, то он закрывает собой остальные элементы документа, так что даже отследить позицию принимающего элемента пользователю будет трудно. С этой проблемой можно справиться, предоставив функцию в качестве значения опции `helper`, как показано в листинге 23.15.

Листинг 23.15. Использование опции `helper`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('div.draggable').draggable({
            helper: function() {
                return $('<img src=lily.png />')
            }
        });

        $('#basket').droppable({
            activeClass: "active",
            hoverClass: "hover"
        });
    });
</script>
...
```

Когда пользователь начинает перетаскивать элемент, jQuery UI вызывает функцию, заданную параметром `helper`, и использует возвращаемый элемент в качестве перемещаемого объекта. В данном случае я использую jQuery для создания элемента `img`. Результат представлен на рис. 23.14.

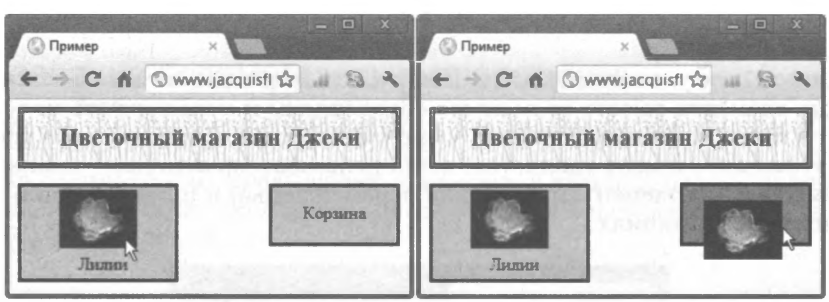


Рис. 23.14. Использование опции `helper`

Небольшое изображение играет роль заместителя перемещаемого элемента, что значительно упрощает отслеживание других элементов документа

Манипуляции вспомогательным элементом

Объект `ui`, который jQuery UI передает событиям взаимодействия Droppable, содержит свойство `helper`, и это свойство можно использовать для манипуляций вспомогательным элементом в процессе его перетаскивания. Пример использования этого свойства в связи с событиями `over` и `out` приведен в листинге 23.16.

Листинг 23.16. Использование свойства `ui.helper`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('div.draggable').draggable({
            helper: function() {
                return $('<img src=lily.png />')
            }
        });

        $('#basket').droppable({
             over: function(event, ui) {
                ui.helper.css("border", "thick solid salmon")
            },
            out: function(event, ui) {
                ui.helper.css("border", "")
            }
        });
    });
</script>
...
```

Здесь события `over` и `out`, а также свойство `ui.helper` используются для отображения рамки вокруг вспомогательного элемента, когда он перекрывает принимающий элемент. Результат представлен на рис. 23.15.

Рис. 23.15. Манипуляции вспомогательным элементом

Привязка к краям элементов

С помощью опции `snap` можно добиться того, чтобы перемещаемый элемент как бы "притягивался" к краям элементов, рядом с которыми он проходит. В качестве значения эта опция принимает селектор. Перемещаемый элемент будет привязываться к краям любого элемента, соответствующего указанному селектору. Пример использования опции `snap` приведен в листинге 23.17.

Листинг 23.17. Использование опции `snap`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
```

```
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #snapper, .draggable, .droppable {font-size: large;
            border: medium solid black;
            padding: 4px; width: 150px; text-align: center;
            background-color: lightgray; margin-bottom: 10px;}
        .droppable {margin-right: 5px; height: 50px; width: 120px}
        #dropContainer {position: absolute; right: 5px;}
        div span {position: relative; top: 25%}
        .droppable.active {border: medium solid black}
        .droppable.hover {background-color: salmon}
        #snapper {position: absolute; left: 35%;
        border: medium solid black; width: 180px; height: 50px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('div.draggable').draggable({
                snap: "#snapper, .droppable",
                snapMode: "both",
                snapTolerance: 50
            });

            $('#basket').droppable({
                activeClass: "active",
                hoverClass: "hover"
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="dropContainer">
        <div id="basket" class="droppable">
            <span>Корзина</span>
        </div>
    </div>
    <div id="snapper"><span>Привяжись здесь</span></div>
    <div class="draggable">
        <span>Перетащи меня</span>
    </div>
</body>
</html>
```

Когда перемещаемый элемент приближается к одному из подходящих элементов, он как бы "притягивается" к нему таким образом, что их соседние края соприкасаются. Для такой привязки можно выбрать любой элемент, а не только принимающий. В этом примере я добавил элемент `div` и определил для опции `snap` значение, которое выбирает в документе данный элемент, а также принимающий элемент. Продемонстрировать описанный эффект привязки с помощью экранных снимков практически невозможно, поэтому я рекомендую читателям самостоятельно поэкспериментировать с этим примером в браузере.

Существует пара вспомогательных опций, позволяющих более точно настроить поведение элементов в отношении привязки. Одна из них — это опция `snapMode`. С ее помощью можно указать тип привязки. Допускаются следующие значения: `inner` (привязка к внутренним краям элементов), `outer` (привязка к внешним краям элементов) и `both` (привязка ко всем краям; используется по умолчанию).

Опция `snapTolerance` позволяет указать, на какое расстояние должен приблизиться перемещаемый элемент к краю элемента-мишени, прежде чем произойдет привязка. Значение по умолчанию — `20`, что означает 20 пикселей. В примере используется значение `50`, которому соответствует привязка на большем расстоянии. Очень важно правильно выбрать значение этой опции. Если значение опции `snapTolerance` слишком мало, то пользователь может не заметить эффекта привязки, а если оно слишком велико, то перемещаемый элемент начнет совершать неожиданные скачки, привязываясь к далеко расположенным элементам.

Резюме

В этой главе вы познакомились с наиболее важными и полезными видами интерактивного взаимодействия jQuery UI — Draggable и Droppable. Здесь было показано, как применять и настраивать каждое из этих взаимодействий по отдельности, как реагировать на связанные с ними события и как настроить их совместную работу для удовлетворения потребностей пользователей веб-приложения в конкретных условиях.

ГЛАВА 24

Использование других взаимодействий

В этой главе описаны оставшиеся три вида взаимодействия jQuery UI: Sortable, Selectable и Resizable. Они менее популярны (и менее полезны) по сравнению со взаимодействиями Draggable и Droppable, описанными в главе 23. Хоть они и находят определенное применение, модели, на которых они основаны, трудно объяснить пользователям. В силу указанных причин лучше всего использовать их как дополнение к другим, более удобным подходам. Перечень тем, рассматриваемых в данной главе, приведен в табл. 24.1.

Таблица 24.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Применение взаимодействия Sortable	Выберите контейнерный элемент и вызовите метод `sortable()`	1
Определение порядка расположения элементов, созданного пользователем с помощью взаимодействия Sortable	Вызовите метод `toArray()` или `serialize()`	2, 3
Разрешение перетаскивания элементов из одного сортируемого элемента в другой	Используйте опцию `connectWith`	4
Связывание перемещаемого элемента с сортируемым элементом	Используйте опцию `connectTo Sortable` при создании перемещаемого элемента	5
Определение того, какие элементы являются сортируемыми	Используйте опцию `items`	6
Стилевое оформление пустого пространства, созданного при перетаскивании сортируемого элемента	Используйте опцию `placeholder`	7
Игнорирование изменения порядка элементов	Используйте метод `cancel()`	8
Обновление набора элементов в сортируемом элементе	Используйте метод `refresh()`	9
Получение информации о выполняющейся операции сортировки	Используйте объект `ui`, предоставляемый обработчикам событий	10
Применение взаимодействия Selectable	Выберите контейнерный элемент и вызовите метод `selectable()`	11, 12
Предотвращение возможности выбора элемента	Используйте метод `cancel()`	13
Применение взаимодействия Resizable	Используйте метод `resizable()`	14

Окончание табл. 24.1

Задача	Решение	Листинг
Одновременное изменение размеров нескольких элементов	Используйте метод `alsoResize()`	15, 16
Ограничение размеров растягиваемых элементов	Используйте опции `maxHeight`, `maxWidth`, `minHeight` и `minWidth`	17
Выбор перемещаемых краев и углов в растягиваемом элементе	Используйте опцию `handles`	18

Использование взаимодействия Sortable

Взаимодействие Sortable позволяет изменять порядок расположения элементов в наборе путем их перетаскивания из одного места в другое. Чтобы применить взаимодействие Sortable, следует выбрать элемент, содержащий отдельные объекты, которые вы хотите отсортировать, и вызвать метод `sortable()`. Соответствующий простой пример приведен в листинге 24.1.

Листинг 24.1. Использование взаимодействия Sortable

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable { width: 100px; background-color: lightgrey;
            font-size: large; float: left; margin: 4px;
            text-align: center; border: medium solid black;
            padding: 4px;}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#sortContainer').sortable();
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="sortContainer">
        <div id="item1" class="sortable">Элемент 1</div>
        <div id="item2" class="sortable">Элемент 2</div>
        <div id="item3" class="sortable">Элемент 3</div>
    </div>
</body>
</html>
```

В этом примере мы создаем ряд элементов div и назначаем им класс sortable. Для создания взаимодействия мы выбираем родительский элемент div (атрибут id которого равен sortContainer) и вызываем метод sortable(). В результате мы получаем возможность менять порядок расположения трех элементов div путем их перетаскивания в новые позиции. Этот процесс проиллюстрирован на рис. 24.1 (однако, как и в случае всех остальных примеров данной главы, будет лучше, если вы запустите данный пример в браузере).

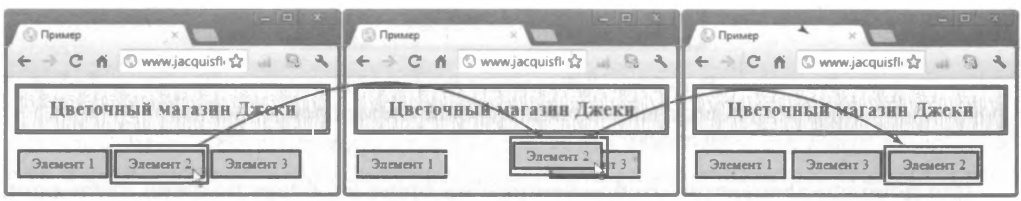


Рис. 24.1. Сортировка элементов путем их перетаскивания

Здесь для демонстрации взаимодействия Sortable элемент под названием Элемент 2 перетаскивается вправо в окне браузера. Как только он минует элемент под названием Элемент 3, элементы переставляются и располагаются в новом порядке. В данном случае элемент перемещался на одну позицию, но ничто не мешает перетаскивать элементы сразу на несколько позиций.

Определение порядка сортируемых элементов

В некоторых случаях требуется определить, в каком порядке располагаются элементы после их перетаскивания пользователем. Для получения этой информации можно вызвать метод toArray, который возвращает JavaScript-массив, содержащий значения атрибута id сортируемых элементов. В листинге 24.2 приведен пример вывода на консоль текущего порядка элементов после щелчка на кнопке.

Листинг 24.2. Получение порядка элементов

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#sortContainer').sortable();

        $('<div id=buttonDiv><button>Получить порядок</button>
            </div>').appendTo('body');
        $('button').button().click(function() {
            var order = $('#sortContainer').sortable("toArray");
            for (var i = 0; i < order.length; i++) {
                console.log("Позиция: " + i + " ID: " + order[i]);
            }
        })
    });
</script>
...
```

Результат представлен на рис. 24.2. В результате щелчка на кнопке вызывается метод toArray, и содержимое результирующего массива выводится на консоль.

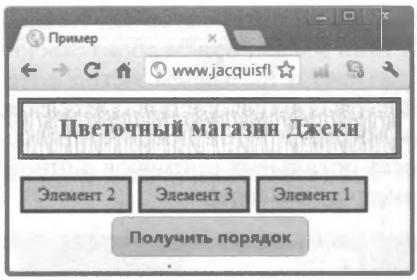


Рис. 24.2. Добавление кнопки для вывода на консоль позиций элементов в списке

Для порядка элементов, отображенного на рисунке, будет получен следующий консольный вывод.

```
Позиция: 0 ID: Элемент 2
Позиция: 1 ID: Элемент 3
Позиция: 2 ID: Элемент 1
```

Кроме того, можно использовать метод `serialize` для генерации строки, которую будет удобно использовать в форме. Соответствующий пример приведен в листинге 24.3.

Листинг 24.3. Использование метода `serialize`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
       .div.sortable { width: 100px; background-color: lightgrey;
            font-size: large; float: left; margin: 4px;
            text-align: center; border: medium solid black;
            padding: 4px;}
            #buttonDiv {clear: both}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#sortContainer').sortable();

            $('<div id=buttonDiv><button>Получить порядок</button>
                </div>').appendTo('body');
            $('button').button().click(function() {
                var formstring = $('#sortContainer')
                    .sortable("serialize");
                console.log(formstring);
            })
```

```
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="sortContainer">
        <div id="item_1" class="sortable">Элемент 1</div>
        <div id="item_2" class="sortable">Элемент 2</div>
        <div id="item_3" class="sortable">Элемент 3</div>
    </div>
</body>
</html>
```

Обратите внимание на то, что мне пришлось изменить значения id сортируемых элементов. При генерации строк метод serialize ищет значения этого атрибута в форме *ключ_индекс*. Для порядка расположения элементов, отображенного на рис. 24.2, будет получен следующий консольный вывод.

```
item[]=2&item[]=3&item[]=1
```

Настройка взаимодействия Sortable

Взаимодействие Sortable в значительной мере зависит от взаимодействия Draggable, описанного в главе 23. Это означает, что все опции взаимодействия Draggable (такие, как axis или tolerance) с тем же эффектом могут применяться и для настройки взаимодействия Sortable. В связи с этим я не буду вновь подробно описывать все настройки и остановлюсь лишь на тех из них, которые свойственны лишь взаимодействию Sortable и чаще всего используются. Их перечень приведен в табл. 24.2, а подробные описания содержатся в следующих разделах.

Таблица 24.2. Свойства взаимодействия Draggable

Свойство	*Описание*
connectWith	Определяет другой сортируемый элемент-контейнер, с которым должна быть установлена связь, обеспечивающая возможность взаимного перемещения элементов между контейнерами. Значение по умолчанию — false; ему соответствует отсутствие таких связей
dropOnEmpty	Если эта опция равна false, то элементы не могут быть перемещены в связанный сортируемый контейнер, когда он пуст. Значение по умолчанию — true
items	Определяет селектор, устанавливающий, какие элементы будут сортируемыми. Значение по умолчанию — > *; оно соответствует выбору всех потомков элемента, для которого был вызван метод sortable()
placeholder	Определяет класс, который будет назначен элементу, созданному для заполнения позиции, занимаемой сортируемым элементом до его перемещения в новое расположение

Связывание сортируемых контейнеров между собой

В средствах сортировки, предоставляемых подключаемым модулем jQuery UI, мне больше всего нравится возможность связывания между собой двух контейнеров, наделенных функциональностью взаимодействия Sortable, что позволяет перемещать элементы из одного контейнера в другой. Это достигается с помощью опции connectWith, используемой для задания селектора, выбирающего элемент, с которым должна быть установлена такая связь. Определив значения свойства

connectWith для обоих элементов, можно сделать эту связь двухсторонней, как показано в листинге 24.4.

Листинг 24.4. Установление связи между взаимодействиями Sortable

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable { width: 100px; background-color: lightgrey;
            font-size: large; margin: 4px; text-align: center;
            border: medium solid black; padding: 4px;}
        #fruitContainer {position: absolute; right:50px}
        #flowerContainer {position: absolute; left:50px}
        div.flower {background-color: salmon}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#fruitContainer').sortable({
                connectWith: '#flowerContainer'
            });
            $('#flowerContainer').sortable({
                connectWith: '#fruitContainer'
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="fruitContainer" class="sortContainer">
        <div id="fruit_1" class="sortable fruit">Яблоки</div>
        <div id="fruit_2" class="sortable fruit">Апельсины</div>
        <div id="fruit_3" class="sortable fruit">Бананы</div>
        <div id="fruit_4" class="sortable fruit">Груши</div>
    </div>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable flower">Астры</div>
        <div id="flower_2" class="sortable flower">Пионы</div>
        <div id="flower_3" class="sortable flower">Лилии</div>
        <div id="flower_4" class="sortable flower">Орхидеи</div>
    </div>
</body>
</html>
```

В этом примере создаются две группы элементов, и для контейнерного элемента каждой группы вызывается метод sortable(). Для связывания групп между собой используется опция connectWith. Результат представлен на рис. 24.3.

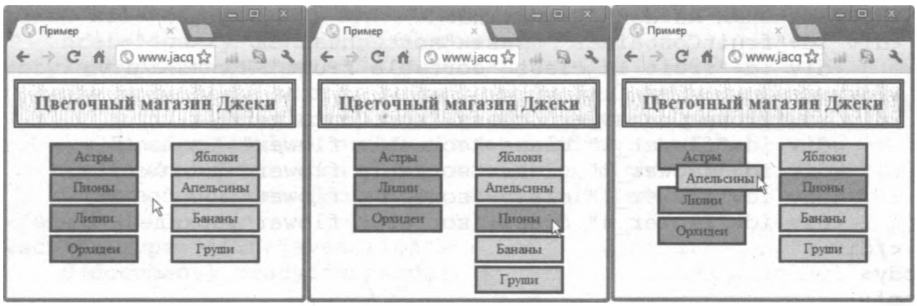


Рис. 24.3. Перемещение элементов между связанными сортируемыми контейнерами

Связывание перемещаемого элемента с сортируемым контейнером

Также имеется возможность связать между собой перемещаемый и сортируемый элементы. Для этого следует использовать в перемещаемом элементе опцию `connectToSortable`, указав в качестве ее значения селектор, выбирающий сортируемый контейнерный элемент, с которым вы хотите установить связь. В листинге 24.5 показано, как это можно сделать.

Листинг 24.5. Связывание перемещаемого элемента с сортируемым контейнерным элементом

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable { width: 100px; background-color: lightgrey;
            font-size: large; margin: 4px; text-align: center;
            border: medium solid black; padding: 4px;}
        #fruitContainer {position: absolute; right:50px}
        #flowerContainer {position: absolute; left:50px}
        div.flower {background-color: salmon}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#fruit_1').draggable({
                connectToSortable: '#flowerContainer',
                helper: "clone"
            });
            $('#flowerContainer').sortable();
        });
    </script>
</head>
<body>
```

```
    <h1>Цветочный магазин Джеки</h1>
    <div id="fruitContainer" class="sortContainer">
        <div id="fruit_1" class="sortable fruit">Яблоки</div>
    </div>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable flower">Астры</div>
        <div id="flower_2" class="sortable flower">Пионы</div>
        <div id="flower_3" class="sortable flower">Лилии</div>
        <div id="flower_4" class="sortable flower">Орхидеи</div>
    </div>
</body>
</html>
```

В этом примере количество элементов в списке фруктов уменьшено до одного, который сделан перемещаемым и связан со списком цветов. Тем самым обеспечена возможность добавления перемещаемого элемента в сортируемый контейнер, как показано на рис. 24.4. Это работает безукоризненно в тех случаях, когда значением свойства `helper` перемещаемого элемента является `clone`. При других значениях этого свойства также получаются правильные результаты, но при этом выводятся сообщения об ошибках.

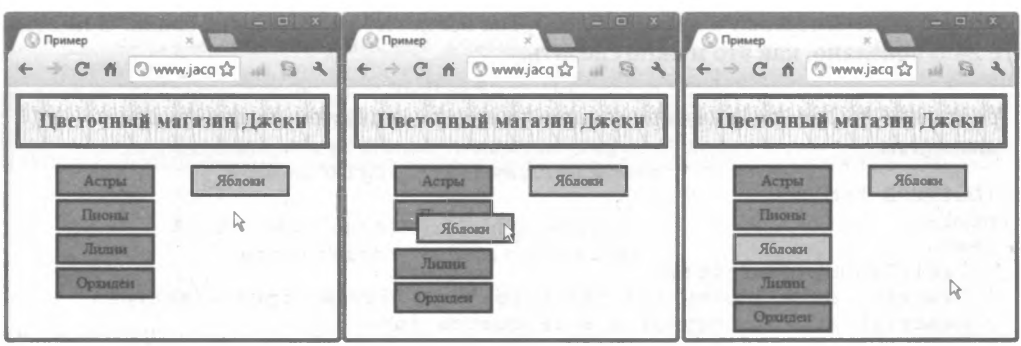


Рис. 24.4. Связывание перемещаемых и сортируемых элементов

Выбор сортируемых элементов

Можно выбирать по своему усмотрению, какие из элементов, содержащихся в контейнере, могут участвовать в сортировке. Для этого используется опция `items`, принимающая в качестве значения селектор элементов, которые вы хотите сделать сортируемыми. Элементы, которые не соответствуют селекторам, не могут переставляться в сортируемом контейнере. Соответствующий пример приведен в листинге 24.6.

Листинг 24.6. Отбор элементов, участвующих в сортировке

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
```

```
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable { width: 100px; background-color: salmon;
            font-size: large; margin: 4px; text-align: center;
            border: medium solid black; padding: 4px;}
        #fruitContainer {position: absolute; right:50px}
        #flowerContainer {position: absolute; left:50px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('div.flower:even').css("background-color", "salmon")

            $('#flowerContainer').sortable({
                items: '.flower:even'
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable flower">Астры</div>
        <div id="flower_2" class="sortable flower">Пионы</div>
        <div id="flower_3" class="sortable flower">Лилии</div>
        <div id="flower_4" class="sortable flower">Орхидеи</div>
    </div>
</body>
</html>
```

В этом примере с помощью опции `items` определяется, что сортируемыми должны быть только четные элементы, содержащиеся в контейнере. Из представленных на рис. 24.5 элементов сортируемыми являются лишь элементы `Астры` и `Лилии`, тогда как элементы `Пионы` и `Орхидеи` не будут реагировать на попытки их перемещения.

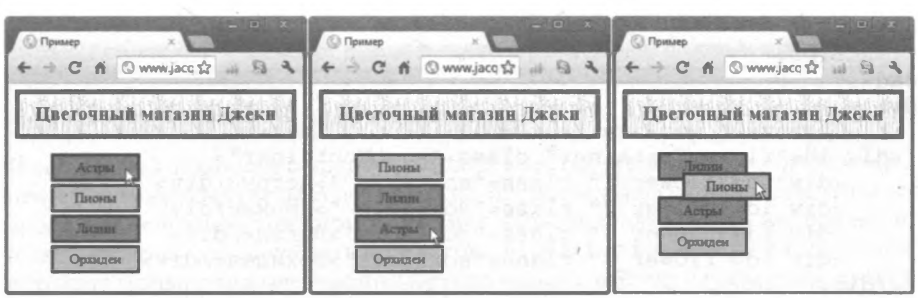


Рис. 24.5. Отбор элементов, которые могут участвовать в сортировке

Работая с опцией `items`, вы должны знать об одной особенности, которую на рисунке отображает последний снимок. Элемент, не соответствующий селектору, нельзя перетащить в новую позицию, если только перед этим он не был вытеснен со своей позиции другим элементом. Так, на приведенном рисунке элемент `Астры` смещается в другую позицию и при этом принудительно смещает элемент `Пионы`. Будучи один раз вытесненным из своей начальной позиции, элемент `Пионы` приоб-

ретает способность к перемещению и сортировке, как если бы он соответствовал селектору, определяемому опцией `items`.

Стилевое оформление опустевшей позиции

В результате перетаскивания элемента в другое место он оставляет после себя незаполненную позицию. Опция `placeholder` позволяет назначить пустой позиции некий класс CSS. Эту возможность удобно использовать для визуального выделения того места в документе, которое готово принять элемент. Пример использования опции `placeholder` приведен в листинге 24.7.

Листинг 24.7. Использование опции `placeholder`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable {width: 100px; background-color: lightgrey;
            font-size: large; margin: 4px; text-align: center;
            border: medium solid black; padding: 4px;}
        #flowerContainer {position: absolute; left:25%}
        .emptySpace {border: medium dotted red; height: 25px;
        margin: 4px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#flowerContainer').sortable({
                placeholder: 'emptySpace'
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable ">Астры</div>
        <div id="flower_2" class="sortable ">Пионы</div>
        <div id="flower_3" class="sortable">Лилии</div>
        <div id="flower_4" class="sortable">Орхидеи</div>
    </div>
</body>
</html>
```

В этом примере я определил класс `emptySpace`, задающий высоту и размер полей, а также пунктирную границу красного цвета для элементов, к которым он применяется. Этот класс задается в качестве значения опции `placeholder`, и, как показано на рис. 24.6, когда элемент перетаскивается, оставленному им пустому пространству присваивается класс `emptySpace`.

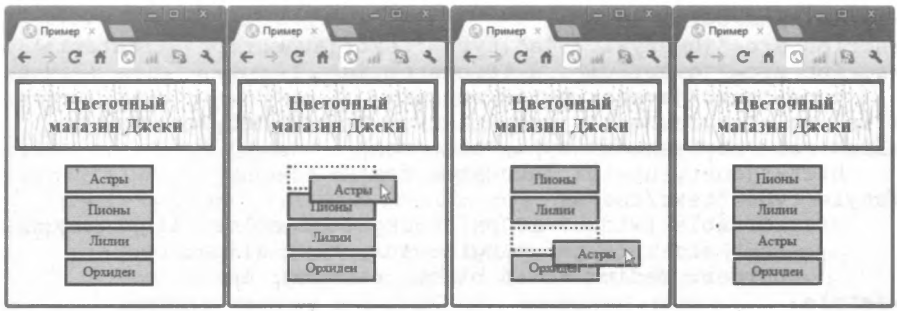


Рис. 24.6. Использование параметра `placeholder`

Использование методов взаимодействия Sortable

Для взаимодействия Sortable определены все стандартные методы jQuery UI плюс несколько дополнительных, являющихся специфическими для работы с сортируемыми элементами. Эти методы перечислены в табл. 24.3.

Таблица 24.3. Методы взаимодействия Sortable

Метод	*Описание*
`sortable("destroy")`	Полностью удаляет всю функциональность взаимодействия Sortable из элемента
`sortable("disable")`	Временно отключает функциональность взаимодействия Sortable для базового элемента
`sortable("enable")`	Включает ранее отключенную функциональность взаимодействия Sortable для базового элемента
`sortable("option")`	Позволяет получить или изменить значение одной или нескольких опций
`sortable("toArray")`	Возвращает массив, содержащий упорядоченный список значений атрибута `id` (см. приведенный ранее пример использования этого метода)
`sortable("refresh")`	Обновляет состояние кеша взаимодействия Sortable
`sortable("cancel")`	Отменяет результат применения последней операции сортировки

Отмена результата последней сортировки

Метод `cancel` позволяет предотвращать участие элементов в сортировке. Этой возможностью не следует злоупотреблять, поскольку фактически она означает игнорирование действий, предпринимаемых пользователем. Если вы все же применяете метод `cancel`, то позаботьтесь о том, чтобы пользователь знал, почему так происходит. Пример использования метода `cancel` вместе с событием `update` приведен в листинге 24.8. Событие `update` происходит тогда, когда пользователь отпускает кнопку мыши, перетащив элемент. События взаимодействия Sortable будут описаны далее.

Листинг 24.8. Использование метода `cancel`

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable {width: 100px; background-color: lightgrey;
            font-size: large; margin: 4px; text-align: center;
            border: medium solid black; padding: 4px;}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#error').dialog({autoOpen: false, modal: true})

            $('#flowerContainer').sortable({
                update: function() {
                    var sortedItems = $('#flowerContainer')
                        .sortable("toArray");
                    if (sortedItems[0] != "item_1") {
                        $('#error').dialog("open")
                        $('#flowerContainer').sortable("cancel")
                    }
                }
            });
        });
    </script>
</head>
<body>
    <div id='error'>На первом месте должен быть "Король"</div>
    <h1>Цветочный магазин Джеки</h1>
    <div id="flowerContainer" class="sortContainer">
        <div id="item_1" class="sortable ">Король</div>
        <div id="item_2" class="sortable ">Королева</div>
        <div id="item_3" class="sortable ">Валет</div>
        <div id="item_4" class="sortable">10</div>
    </div>
</body>
</html>
```

В этом примере метод `cancel` вызывается в том случае, если в новом порядке расположения элементов, созданном пользователем, элемент `Король` не находится на первом месте. Для уведомления пользователя о возникших проблемах используется виджет Dialog, описанный в главе 22. Изменениям, затрагивающим порядок расположения других элементов, разрешается вступить в силу.

Обновление сортируемых элементов

Вызов метода `refresh` заставляет взаимодействие Sortable обновить свой кеш элементов, содержащихся в сортируемом контейнере. Пример использования этого метода для добавления новых сортируемых элементов приведен в листинге 24.9.

Листинг 24.9. Добавление новых сортируемых элементов

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable {width: 100px; background-color: lightgrey;
            font-size: large; margin: 4px; text-align: center;
            border: medium solid black; padding: 4px;}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('#flowerContainer').sortable();

            var itemCount = 2;

            $('button').click(function() {
                $('<div id=flower_' + (itemCount++) +
                    'class=sortable>Элемент ' + itemCount +
                    '</div>').appendTo('#flowerContainer');
                $('#flowerContainer').sortable("refresh");
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <button>Добавить сортируемый элемент</button>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable">Астры</div>
        <div id="flower_2" class="sortable">Пионы</div>
    </div>
</body>
</html>
```

В этом примере в документ добавлена кнопка, с помощью которой в сортируемый контейнер можно добавлять новые элементы, а вызов метода `refresh` гарантирует корректное распознавание нового порядка элементов.

Использование событий взаимодействия Sortable

Взаимодействие Sortable поддерживает все события, определенные для взаимодействия Draggable, которые были описаны в главе 23. Кроме того, взаимодействие Sortable поддерживает ряд собственных событий, перечень которых приведен в табл. 24.4.

Таблица 24.4. События взаимодействия Sortable

Событие	*Описание*
`change`	Происходит при изменении позиции элемента в результате сортировки, выполненной пользователем
`receive`	Происходит при перемещении элемента в данный сортируемый элемент-контейнер из другого связанного сортируемого элемента-контейнера

Окончание табл. 24.4

Событие	Описание
remove	Происходит при перемещении элемента из данного сортируемого элемента-контейнера в другой связанный сортируемый элемент-контейнер
sort	Происходит при каждом перемещении мыши в процессе сортировки
update	Происходит при завершении перемещения элемента пользователем при условии, что порядок элементов был изменен

При наступлении каждого из этих событий jQuery UI предоставляет дополнительную информацию посредством передаваемого обработчику события в качестве аргумента объекта `ui`, свойства которого перечислены в табл. 24.5.

Таблица 24.5. Свойства объекта `ui` взаимодействия Sortable

Свойство	Описание
helper	Возвращает вспомогательный элемент
position	Возвращает информацию о текущем местоположении вспомогательного элемента в виде объекта со свойствами `top` и `left`
item	Возвращает объект `jQuery`, содержащий перемещаемый элемент
placeholder	Возвращает объект `jQuery`, представляющий позицию, с которой был перемещен или куда будет перемещен сортируемый элемент
sender	Возвращает объект `jQuery`, содержащий связанный сортируемый контейнерный элемент, в котором ранее находился перемещенный элемент (в отсутствие связанных сортируемых контейнеров значение этого свойства равно `null`)

Пример использования объекта `ui` вместе с событиями `sort` и `change` приведен в листинге 24.10.

Листинг 24.10. Использование событий `sort` и `change`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        div.sortable {width: 100px; background-color: lightgrey;
        font-size: large; margin: 4px; text-align: center;
        border: medium solid black; padding: 4px;}
        #flowerContainer {position: absolute; left:10px}
        #info {position: absolute; right: 10px;
            border: medium solid black; padding: 4px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
```

```
            $('#flowerContainer').sortable({
                sort: function(event, ui) {
                    $('#itemId').text(ui.item.attr("id"))
                },
                change: function(event, ui) {
                    $('#pos').text($('#flowerContainer *')
                        .index(ui.placeholder))
                }
            });
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable ">Астры</div>
        <div id="flower_2" class="sortable ">Пионы</div>
        <div id="flower_3" class="sortable">Лилии</div>
        <div id="flower_4" class="sortable">Орхидеи</div>
    </div>
    <div id="info" class="ui-widget">
        <div>ID элемента: <span id="itemId">Не определено</span>
        </div>
        <div>Позиция: <span id="pos">Не определено</span></div>
    </div>
</body>
</html>
```

Здесь события используются для отображения информации о выполняемой операции сортировки. Функция — обработчик события `sort` считывает значение свойства `ui.item` и получает значение атрибута `id` перемещаемого элемента. Обработчик события `change` считывает значение свойства `ui.placeholder` и использует метод `index` для вычисления позиции заместителя элемента среди сортируемых элементов.

Использование взаимодействия Selectable

С помощью взаимодействия Selectable пользователь может выбирать один или несколько элементов путем перемещения указателя мыши или выполнения щелчков на отдельных элементах. Для применения этого вида взаимодействия следует вызвать метод `selectable()`, как показано в листинге 24.11.

Листинг 24.11. Применение взаимодействия Selectable

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
```

```
        div.flower {width: 200px; background-color: lightgrey;
            font-size: large; margin: 4px; text-align: center;
            border: medium solid black; padding: 4px;}
        #flowerContainer {position: absolute; left:10px}
        div.ui-selected {border: medium solid cyan;
            background-color: salmon}
        div.ui-selecting {border: medium solid salmon}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#flowerContainer').selectable();
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="flowerContainer">
        <div id="flower_1" class="flower">Астры</div>
        <div id="flower_2" class="flower">Пионы</div>
        <div id="flower_3" class="flower">Лилии</div>
        <div id="flower_4" class="flower">Орхидеи</div>
    </div>
</body>
</html>
```

Взаимодействие Selectable применяется к элементу, содержащему те элементы, возможность выбора которых вы хотите предоставить пользователю. В данном случае используются те же элементы `div`, что и при рассмотрении взаимодействия Sortable. Мы выбираем элемент-контейнер и вызываем для него метод `selectable()`.

```
$('#flowerContainer').selectable();
```

Контейнер обладает функциональностью взаимодействия Selectable, однако нам еще остается определить пару стилей для некоторых специальных классов, обеспечивающих визуальную обратную связь с пользователем. Вот эти стили.

```
div.ui-selected {border: medium solid cyan;
    background-color: salmon}
div.ui-selecting {border: medium solid salmon}
```

Взаимодействие Selectable применяет эти классы к элементам для визуальной индикации состояния их выбора. Класс `ui-selecting` применяется в тех случаях, когда пользователь перемещает мышь для выбора элементов, расположенных в определенной области, а класс `ui-selected` — когда элемент оказывается выбранным (либо в результате выполнения на нем щелчка, либо потому, что он оказался в области, охваченной указателем мыши при ее перемещении). В примере использованы простые стили, которые всего лишь изменяют цвет фона и добавляют границы (рамки). Результат выбора элементов путем перемещения указателя мыши представлен на рис. 24.7.

Приступая к выбору элементов, пользователь должен начинать перемещение мыши внутри контейнерного элемента. На среднем из приведенных на рисунке снимков вы видите контур выбранной области (называемый *рамкой выбора*); в этот момент jQuery UI применяет класс `ui-selecting`. При отпускании кнопки мыши элементы, охватываемые (полностью или частично) рамкой, становятся выбранными, и к ним применяется класс `ui-selecting`, как показано на последнем снимке.

Пользователи также могут выбирать элементы с помощью щелчков. Чтобы выбрать несколько несмежных элементов посредством щелчков мыши, следует одновременно удерживать в нажатом состоянии клавишу <Ctrl> (<Meta>). Если щелчки

приводят лишь к переключению состояния единственного выбранного элемента, добавьте в сценарий код, выделенный в листинге 24.12 полужирным шрифтом.

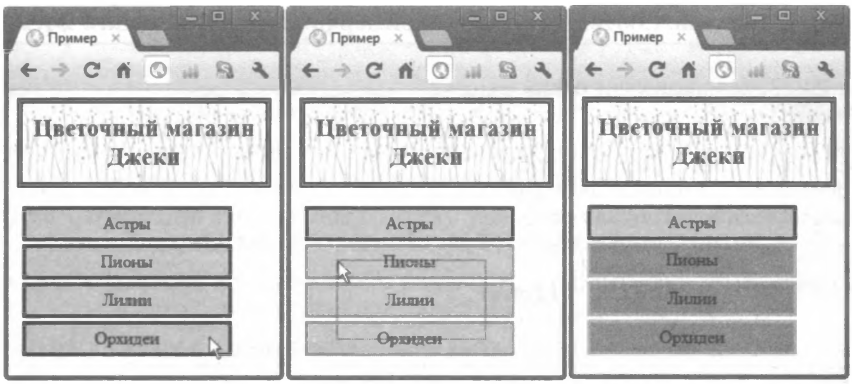


Рис. 24.7. Выбор элементов с помощью мыши

Листинг 24.12. Разрешение выбора нескольких элементов при использовании взаимодействия Selectable

```
<script type="text/javascript">
    $(document).ready(function() {
        $('#flowerContainer')
            .bind("mousedown", function(e) {e.metaKey = true;})
            .selectable();
    });
</script>
```

В момент нажатия пользователем кнопки мыши применяется класс `ui-selecting`. Класс `ui-selected` применяется при отпускании кнопки мыши.

Настройка взаимодействия Selectable

Для настройки взаимодействия Selectable используются свойства, перечень которых приведен в табл. 24.6.

Таблица 24.6. Свойства взаимодействия Selectable

Свойство	*Описание*
`disabled`	Если значение этой опции равно `true`, то функциональность взаимодействия для данного элемента первоначально отключена. Значение по умолчанию — `false`
`autoRefresh`	Если значение этой опции равно `true`, то в начале каждой операции выбора осуществляется пересчет размеров и положений каждого из выбираемых элементов. Значение по умолчанию — `true`
`cancel`	Строка селектора jQuery, предотвращающего выбор соответствующих элементов
`delay`	См. описание опции `delay` для взаимодействия Draggable в главе 23
`distance`	См. описание опции `distance` для взаимодействия Draggable в главе 23
`filter`	Селектор, используемый для выбора элементов в контейнере, наделенных функциональностью взаимодействия Selectable. Значение по умолчанию — `*`; ему соответствуют все элементы

Смысл большинства этих свойств либо очевиден, либо аналогичен смыслу одноименных свойств других видов взаимодействия. Из них особый интерес представляет свойство `cancel`, которое позволяет предотвратить выбор пользователем некоторых элементов. Соответствующий пример приведен в листинге 24.13.

Листинг 24.13. Использование опции `cancel`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#flowerContainer')
            .bind("mousedown", function(e) {e.metaKey = true;})
            .selectable({
                cancel: '#flower_3'
            );
    });
</script>
...
```

В этом сценарии использован селектор, препятствующий выбору элемента с идентификатором `flower_3`. Этот подход хорошо работает в тех случаях, когда пользователь выбирает элементы с помощью щелчков, но не в состоянии воспрепятствовать выбору элементов путем растягивания вокруг них рамки. Поэтому при использовании опции `cancel` следует учитывать данный факт.

Использование методов взаимодействия Selectable

Как видно из данных, приведенных в табл. 24.7, взаимодействие Selectable имеет только один специфический метод. Другие методы являются общими для всех виджетов и взаимодействий.

Таблица 24.7. Методы взаимодействия Sortable

Метод	*Описание*
`selectable("destroy")`	Полностью удаляет всю функциональность взаимодействия Selectable из базового элемента
`selectable("disable")`	Временно отключает функциональность взаимодействия Selectable для базового элемента
`selectable("enable")`	Включает ранее отключенную функциональность взаимодействия Selectable для базового элемента
`selectable("option")`	Позволяет получить или изменить значение одного или нескольких параметров
`selectable("refresh")`	Обновляет состояние взаимодействия Selectable. Это вариант ручной настройки, аналогичный использованию значения `false` для параметра `autoRefresh`

Использование событий взаимодействия Selectable

События, определенные для взаимодействия Selectable, перечислены в табл. 24.8.

Для большинства перечисленных событий jQuery UI предоставляет дополнительную информацию посредством объекта `ui`. Для событий `selected` и `unselected` в объекте `ui` предусмотрено свойство `selected`, которое содержит объект `HTMLElement`,

представляющий выбранный (или находящийся в процессе выбора) элемент. Для событий `unselected` и `unselecting` предусмотрено свойство `unselected`, которое предназначено для аналогичных целей, но относится к отмене выбора элемента.

Таблица 24.8. События взаимодействия Selectable

Событие	*Описание*
`create`	Происходит в момент применения взаимодействия Selectable к данному элементу
`selected`	Происходит по завершении выбора элемента. Если выбрано несколько элементов, то это событие наступает для каждого из них по отдельности
`selecting`	Происходит в момент начала выбора (путем нажатия кнопки мыши или перемещения указателя мыши)
`unselected`	Происходит при отмене выбора элемента. Если выбор отменен для нескольких элементов, это событие наступает для каждого из них по отдельности
`unselecting`	Происходит в момент начала отмены выбора путем нажатия кнопки мыши

Использование взаимодействия Resizable

Взаимодействие Resizable добавляет в элемент манипуляторы, перемещая которые, пользователь может масштабировать элемент путем изменения его размеров. Некоторые браузеры автоматически предоставляют такую возможность для текстовых областей, но взаимодействие Resizable обеспечивает возможность подобного масштабирования для любого элемента в документе. Пример применения данного вида взаимодействия, которое реализуется с помощью метода `resizable()`, приведен в листинге 24.14.

Листинг 24.14. Применение взаимодействия Resizable

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #astor, #lily {text-align: center; width: 150px;
            border: thin solid black; padding: 5px; float: left;
            margin: 20px}
        #astor img, #lily img {display: block; margin: auto}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#astor').resizable({
                alsoResize: "#astor img"
            });
        });
    </script>
</head>
```

```
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div id="astor" class="ui-widget">
        <img src="astor.png" />
        Астры
    </div>
     <div id="lily" class="ui-widget">
        <img src="lily.png" />
        Лилии
    </div>
</body>
</html>
```

В этом примере создаются два элемента `div`, каждый из которых содержит элемент `img` и текст. В сценарии один из них выбирается, и к нему применяется метод `resizable()` (с использованием параметра `alsoResize`, который будет описан далее). Библиотека jQuery UI добавляет к выбранному элементу манипулятор, позволяющий изменять вертикальный и горизонтальный размеры элемента, как видно на рис. 24.8. На рисунке элемент представлен с увеличенной высотой и уменьшенной шириной.

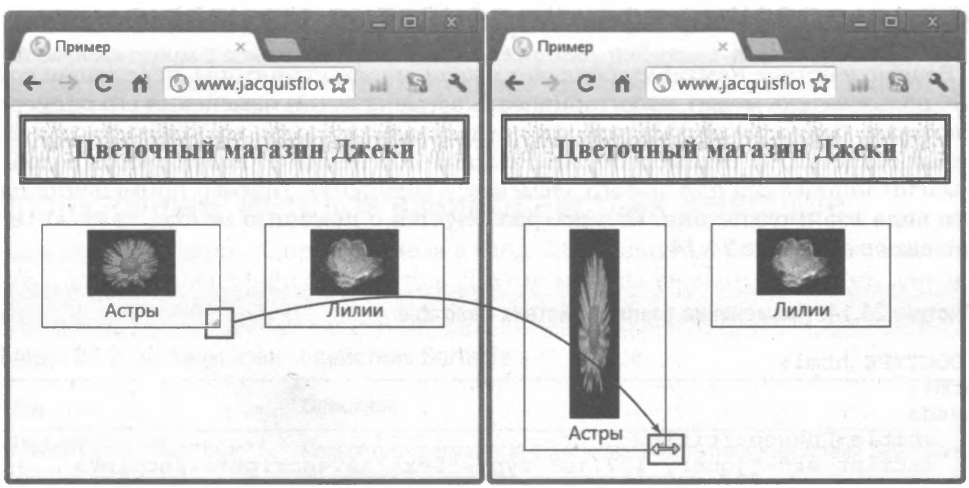


Рис. 24.8. Использование манипулятора для изменения размеров элемента

Настройка взаимодействия Resizable

Для настройки взаимодействия Resizable используются свойства, описанные в табл. 24.9. Взаимодействие Resizable зависит от взаимодействия Draggable, описанного в главе 23. Это означает, что в дополнение к настройкам, приведенным в таблице, можно использовать настройки взаимодействия Draggable, в том числе такие, как `delay`, `distance`, `grid` и `containment`.

Таблица 24.9. Свойства взаимодействия Resizable

Свойство	*Описание*
`alsoResize`	Селектор, используемый для выбора элементов, размеры которых должны изменяться одновременно с размерами элемента, к которому применено взаимодействие Resizable. Значение по умолчанию — `false`; оно означает отсутствие таких элементов

Окончание табл. 24.9

Свойство	Описание
aspectRatio	Если значение этой опции равно true, то изменение размеров элемента будет происходить с сохранением пропорции соотношения сторон. Значение по умолчанию — true
autoHide	Если значение этой опции равно true, то манипуляторы становятся видимыми лишь тогда, когда указатель мыши располагается над элементом. Значение по умолчанию — false
ghost	Если значение этой опции равно true, то при изменении размеров элемента будут видны полупрозрачные контуры, отображающие новые размеры элемента. Значение по умолчанию — true
handles	Определяет, где будут располагаться манипуляторы. Поддерживаемые значения приведены далее
maxHeight	Определяет максимальную высоту, до которой можно изменить размеры элемента. Значение по умолчанию — null; оно означает отсутствие ограничений
maxWidth	Определяет максимальную ширину, до которой можно изменить размеры элемента. Значение по умолчанию — null; оно означает отсутствие ограничений
minHeight	Определяет минимальную высоту, до которой можно изменить размеры элемента. Значение по умолчанию — null; оно означает отсутствие ограничений
minWidth	Определяет минимальную высоту, до которой можно изменить размеры элемента. Значение по умолчанию — null; оно означает отсутствие ограничений

Одновременное изменение размеров других элементов

По моему мнению, наиболее употребительной при настройке взаимодействия Resizable является опция alsoResize. С ее помощью можно определить дополнительные элементы, размеры которых будут изменяться одновременно с размерами элемента, к которому был применен метод resizable(). Я использую эту опцию главным образом для того, чтобы обеспечить синхронное изменение размеров элементов вместе с размерами их родительских элементов. Мы уже использовали эту возможность в предыдущем примере, определив одновременное изменение размеров элементов img и div. Прежде всего, посмотрим, что происходит, если опция alsoResize не используется. Соответствующий код приведен в листинге 24.15.

Листинг 24.15. Изменение размеров элемента, имеющего содержимое, без использования опции alsoResize

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#astor').resizable();
    });
</script>
...
```

Если опция alsoResize не используется, то изменяются только размеры элемента div. Размеры содержащихся в нем элементов остаются неизменными. Что при этом происходит, показано на рис. 24.9.

Иногда именно такой результат и требуется получить, но лично я использую опцию `alsoResize` почти во всех случаях применения взаимодействия Resizable. В этом параметре мне нравится то, что выбор подходящих элементов не ограничивается содержимым того элемента, размеры которого изменяются. С помощью этой опции можно указать любой другой элемент, как показано в листинге 24.16.

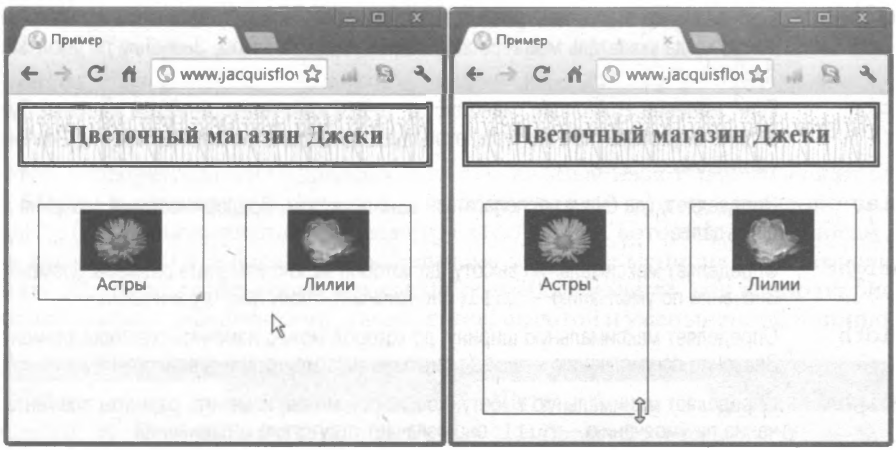


Рис. 24.9. Изменение размеров элемента без изменения размеров его содержимого

Листинг 24.16. Изменение размеров других элементов с помощью опции `alsoResize`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#astor').resizable({
            alsoResize: "#astor img, #lily, #lily img"
        });
    });
</script>
...
```

В этом сценарии выбор элементов расширен с целью включения других элементов `div` и `img`. Таким образом, при изменении размеров одного элемента `div` jQuery UI будет изменять размеры сразу четырех элементов. Результат представлен на рис. 24.10.

Ограничение допустимых пределов для изменения размеров элементов

Можно ограничить пределы изменения размеров масштабируемых элементов, применив опции `maxHeight`, `maxWidth`, `minHeight` и `minWidth`. Значениями этих опций могут быть числа, выражающие количество пикселей, или `null`. Пример использования этих настроек приведен в листинге 24.17.

Совет. По умолчанию для опций `minHeight` и `minWidth` используется значение `10` пикселей. При меньших значениях jQuery UI не сможет отобразить манипуляторы, а это означает, что пользователь не будет в состоянии вновь увеличить размеры элемента. Поэтому меньшие значения следует использовать с осторожностью.

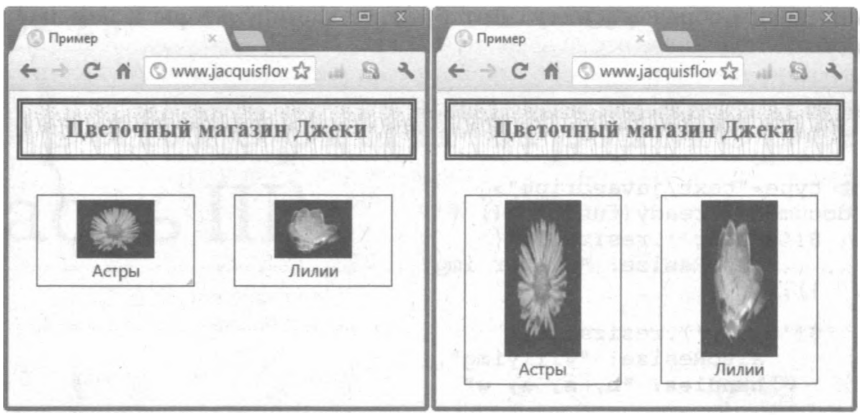


Рис. 24.10. Одновременное изменение размеров нескольких элементов

Листинг 24.17. Ограничение пределов изменения размеров элемента

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#astor').resizable({
            alsoResize: "#astor img",
            maxWidth: 200,
            maxHeight: 150
        });
    });
</script>
...
```

Совет. Можно также использовать опцию `containment`, определенную для взаимодействия Draggable, описанного в главе 23. Это позволит ограничить максимальный размер элемента размером другого элемента.

Размещение манипуляторов

С помощью опции `handles` можно определить, какие края и углы разрешено перемещать для изменения размеров элемента. В качестве значения этой опции можно указать `all` (перемещаемыми являются все края и углы) или любую комбинацию компасных точек (`n`, `e`, `s`, `w`, `ne`, `se`, `nw`, `sw`), определяющих отдельные края и углы.

Можно указать несколько значений, разделив их запятыми. Значением по умолчанию является `e, s, se`, которое означает, что перемещаемыми будут нижний правый угол (`se`), а также правый (`e`) и нижний (`s`) край. Манипулятор диагонального перемещения отображается лишь в нижнем правом углу и только в том случае, если значение опции `handles` содержит `se`. При наведении указателя мыши на остальные края или углы внешний вид указателя будет изменяться, отображая возможные направления перемещения данного края или угла. Пример использования опции `handles` приведен в листинге 24.18.

В этом сценарии к обоим элементам `div` применено взаимодействие Resizable, и для одного из них определен нестандартный набор манипуляторов. На рис. 24.11

показано, каким образом jQuery UI отображает манипуляторы и изменяет внешний вид указателя.

Листинг 24.18. Использование опции `handles`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('#astor').resizable({
            alsoResize: "#astor img"
        });

        $('#lily').resizable({
            alsoResize: "#lilyimg",
            handles: "n, s, e, w"

        });
    });
</script>
...
```

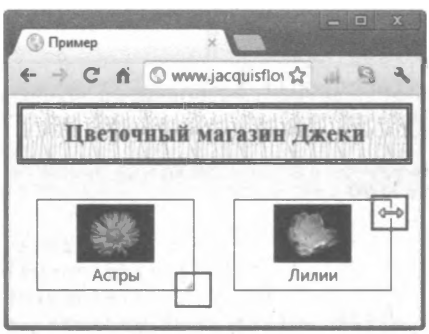


Рис. 24.11. Использование опции `handles`

Резюме

В этой главе вы познакомились с тремя видами взаимодействия jQuery UI: Sortable, Selectable и Resizable. Они менее популярны, чем взаимодействия Draggable и Droppable, описанные в главе 23, но при умелом использовании также могут быть полезными. Как и при организации любых видов интерактивного взаимодействия вообще, основные трудности связаны с тем, что в условиях, когда необходимые визуальные средства индикации не предусмотрены стандартами, приходится самостоятельно продумывать, каким образом дать пользователю понять, что он имеет возможность перемещать, выбирать, сортировать и масштабировать элементы. Поэтому упомянутые виды интерактивной функциональности следует использовать в качестве дополнения к другим механизмам обеспечения интерактивного взаимодействия пользователя с приложением или документом. При таком подходе опытные пользователи смогут распознать предлагаемые новые возможности, тогда как остальные смогут воспользоваться привычными для них очевидными методиками.

ГЛАВА 25

Рефакторинг примера (часть III)

Эта часть книги посвящена виджетам и взаимодействиям jQuery UI, которые позволяют создавать насыщенные веб-приложения со стилевым оформлением пользовательского интерфейса, подчиняющимся определенной теме, и обеспечивают чрезвычайно гибкие возможности настройки в соответствии с конкретными задачами. В данной главе некоторые из указанных средств будут добавлены в наш базовый пример для демонстрации того, каким образом можно организовать их совместную работу.

Дальнейший пересмотр переработанного варианта документа

Ранее в процессе улучшения переработанного варианта базового документа мы, имея в своем распоряжении рассмотренные к тому времени возможности ядра библиотеки jQuery, фактически самостоятельно воспроизводили некоторые из элементов функциональности, которые в готовом виде содержатся в библиотеке jQuery UI. Результаты, которых нам тогда удалось достичь, представлены на рис. 25.1.

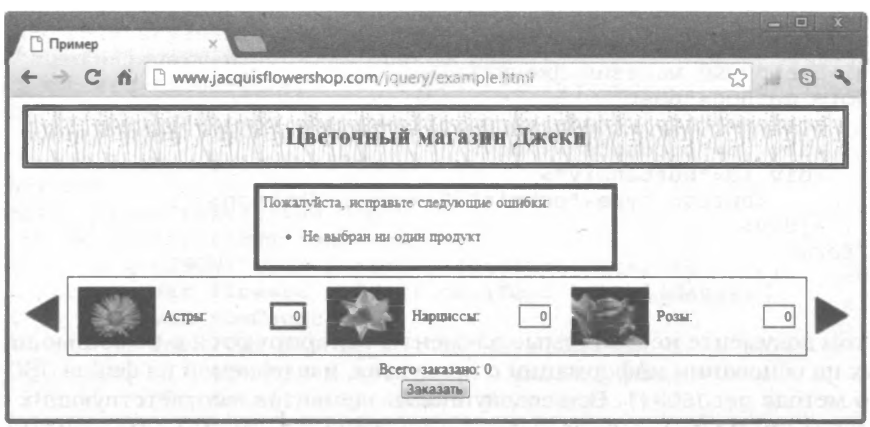


Рис. 25.1. Последний переработанный вариант документа

Среди крупных изменений, которые мы внесли в документ в предыдущей части, можно отметить использование шаблонов данных, проверку корректности данных,

введенных с помощью формы, и выполнение асинхронных запросов средствами Ajax. Мы также организовали "карусельный" способ просмотра предлагаемых продуктов, при котором необходимая информация о них выводится лишь для ограниченного их числа за один раз. Некоторые из упомянутых возможностей будут использованы и в этой главе, но теперь основное внимание будет уделено применению возможностей jQuery UI. В качестве отправной точки для данной главы мы используем вариант документа, приведенный в листинге 25.1.

Листинг 25.1. Начальный вариант документа для данной главы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $.getJSON("mydata.json", function(data) {
                $('#flowerTmpl').tmpl(data).appendTo("#products");
            });
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="products"></div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

В этом документе необходимые элементы генерируются с помощью шаблона данных на основании информации о продуктах, извлекаемой из файла JSON с помощью метода `getJSON()`. Вся совокупность элементов, соответствующих отдельным видам продукции, собирается в единственном элементе с идентификатором `products`. Вид исходного документа в окне браузера представлен на рис. 25.2.

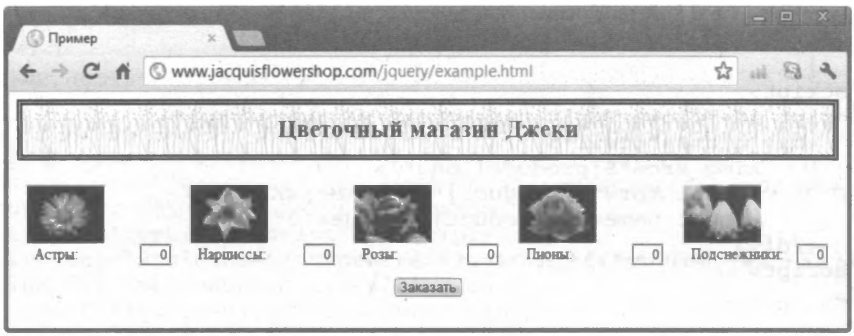


Рис. 25.2. Исходный документ для данной главы

Отображение продуктов

В качестве средства отображения продуктов для пользователя мы воспользуемся виджетом Accordion. Несмотря на то что имеется всего лишь шесть продуктов, мы разобьем их на группы, в каждую из которых войдет по два продукта, а для создания структуры элементов, которая требуется для виджета Accordion, используем jQuery. Соответствующие изменения представлены в листинге 25.2.

Листинг 25.2. Упорядочение и структуризация элементов, соответствующих отдельным видам цветов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <style type="text/css">
        #accordion {margin: 5x}
        .dcell img {height: 60px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $.getJSON("mydata.json", function(data) {
                var flowers = $('#flowerTmpl').tmpl(data);
                var rowCount = 1;
                for (var i = 0; i < flowers.length; i += 2) {
                    $("<h2><a href=#>" + data[i].name + " и " +
                    data[i + 1].name + "</a></h2>")
                    .appendTo("#products");
                    $("<div id='row" + (rowCount++) + "'></div>")
                        .appendTo("#products")
                        .append(flowers.slice(i, i + 2))
                }
```

```
                $('#products').accordion({fillSpace: true});
            });
        });
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="products"></div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Здесь в функцию, передаваемую методу `getJSON()`, добавлен код, предназначенный для создания виджета Accordion, включая построение необходимой структуры элементов и вызов метода `accordion()`. В новой реализации названия цветов извлекаются из соответствующего источника с помощью объекта данных JSON, но для генерации HTML-элементов (которые затем разбиваются на группы и помещаются в оболочки, образуемые элементами `div`, в соответствии с требованиями виджета Accordion) по-прежнему используется подключаемый модуль шаблонов данных. Вид документа в окне браузера до и после добавления вызова метода `accordion()` представлен на рис. 25.3.

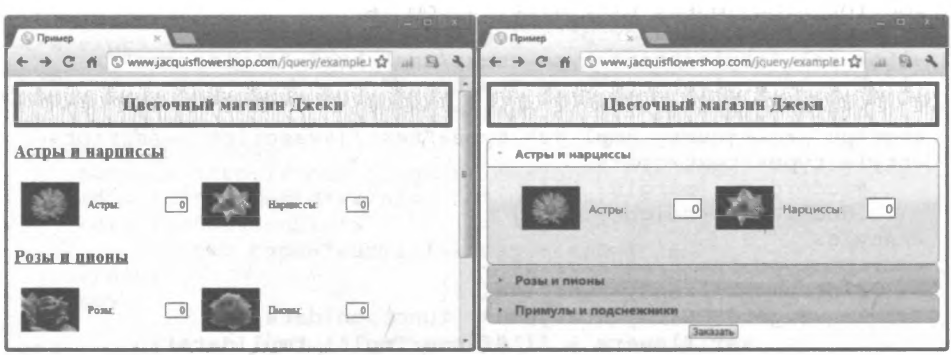


Рис. 25.3. Создание структуры элементов и вызов метода `accordion()`

Добавление корзины покупателя

Нашим следующим шагом будет добавление простейшей корзины покупателя, представляющей отобранные покупателем продукты. Соответствующие изменения, которые для этого требуется внести в образец документа, представлены в листинге 25.3.

Листинг 25.3. Добавление корзины покупателя

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <style type="text/css">
        .dcell img {height: 60px}
        #basketTable {border: thin solid black;
            border-collapse: collapse}
        th, td {padding: 4px; width: 50px}
        td:first-child, th:first-child {width: 150px}
        #placeholder {text-align: center}
        #productWrapper {float: left; width: 65%}
        #basket {width: 30%; text-align: left; float: left;
            margin-left: 10px}
        #buttonDiv {clear: both}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $.getJSON("mydata.json", function(data) {
                var flowers = $('#flowerTmpl').tmpl(data);
                var rowCount = 1;
                for (var i = 0; i < flowers.length; i += 2) {
                    $("<h2><a href=#>" + data[i].name + " & " +
                        data[i + 1].name + "</a></h2>")
                        .appendTo("#products");
                    $("<div id='row" + (rowCount++) + "'></div>")
                        .appendTo("#products")
                        .append(flowers.slice(i, i + 2))
                }
                $('#products').accordion();

                $('input').change(function(event) {
                    $('#placeholder').hide();
                    var fname = $(this).attr("name");
                    var row = $('tr[id=' + fname + ']');
                    if (row.length == 0) {
                        $('#rowTmpl').tmpl({
                            name: fname,
                            val:  $(this).val(),
                            product: $(this).siblings("label")
                                .text()
                        }).appendTo("#basketTable").find("a")
                            .click(function() {
                                removeTableRow($(this)
                                    .closest("tr"));
                                var iElem = $('#products')
                                    .find("input[name=" + fname +
```

```
                                        "]")
                                $('#products')
                                    .accordion("activate", iElem
                                    .closest("div[id^=row]")
                                    .prev())
                                    iElem.val(0).select();
                            })
                        } else if ($(this).val() != "0") {
                            row.children().eq(1).text($(this).val())
                        } else {
                            removeTableRow(row)
                        }
                    })
                });

                function removeTableRow(row) {
                    row.remove();
                    if ($('#basketTable tbody')
                        .children(':visible').length == 1) {
                        $('#placeholder').show();
                    }
                }

            });
        </script>
        <script id="rowTmpl" type="text/x-jquery-tmpl">
            <tr id=${name}><td>${product}</td><td>${val}</td>
                <td><a href=#>Remove</a></td></tr>
        </script>
        <script id="flowerTmpl" type="text/x-jquery-tmpl">
            <div class="dcell">
                <img src="${product}.png"/>
                <label for="${product}">${name}:</label>
                <input name="${product}" value="0" />
            </div>
        </script>
    </head>
    <body>
        <h1>Цветочный магазин Джеки</h1>
        <form method="post"
            action="http://node.jacquisflowershop.com:9999/order">
            <div id="productWrapper">
                <div id="products"></div>
            </div>
            <div id="basket" class="ui-widget">
                <table border=1 id="basketTable">
                    <tr><th>Продукт</th><th>Количество</th>
                        <th>Удалить</th></tr>
                    <tr id="placeholder"><td colspan=3>
                        Продукты не выбраны </td></tr>
                </table>
            </div>
            <div id="buttonDiv">
                <button type="submit">Заказать</button>
            </div>
```

```
    </form>
</body>
</html>
```

Помещение виджета Accordion в оболочку

Мы хотим, чтобы корзина покупателя отображалась рядом с панелями виджета Accordion. Для этого мы помещаем элемент, для которого вызывается метод `accordion()`, внутрь другого элемента `div`.

```
<div id="productWrapper">
    <div id="products"></div>
</div>
```

Работа виджета Accordion будет нарушена, если окажется, что он не занимает все пространство родительского элемента по ширине, поэтому мы добавляем оболочку и фиксируем ее ширину с помощью CSS-свойства `width`.

```
#productWrapper {float: left; width: 65%}
```

Таким образом, виджет Accordion, как и должно быть, благополучно располагается по всей ширине элемента-оболочки `div`, который занимает только 65% ширины своего родительского элемента.

Добавление таблицы

Для отображения корзины мы используем элемент `table`, который включаем в число статических элементов документа.

```
<div id="basket" class="ui-widget">
    <table border=1 id="basketTable">
        <tr>
            <th>Продукт</th><th>Количество</th><th>Удалить</th>
        </tr>
        <tr id="placeholder">
            <td colspan=3> Продукты не выбраны </td>
        </tr>
    </table>
</div>
```

Как и в случае виджета Accordion, мы помещаем элемент `table` в оболочку, ширину которой устанавливаем с помощью CSS-свойства.

```
#basket {width: 30%; text-align: left; float: left;
    margin-left: 10px}
```

В таблице содержится строка, в которой располагаются заголовки столбцов, и ряд-заполнитель, занимающий всю таблицу по ширине. Полученный на данном этапе результат представлен на рис. 25.4.

Обработка изменений входных значений

Чтобы связать таблицу с виджетом Accordion, мы реагируем на события `change`, порождаемые элементами `input`, которые создаются в функции `getJSON()`. Это делается с помощью следующей функции-обработчика.

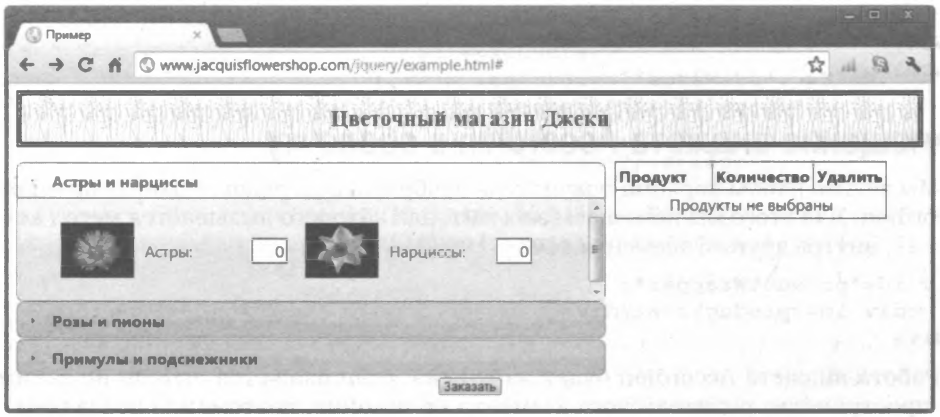


Рис. 25.4. Добавление таблицы в документ

```
$('input').change(function(event) {
    $('#placeholder').hide();
    var fname = $(this).attr("name");
    var row = $('tr[id=' + fname + ']');
    if (row.length == 0) {
        $('#rowTmpl').tmpl({
            name: fname,
            val:  $(this).val(),
            product: $(this).siblings("label")
                .text()
        }).appendTo("#basketTable").find("a")
            .click(function() {
                removeTableRow($(this)
                    .closest("tr"));
                var iElem = $('#products')
                    .find("input[name=" + fname + "]")
                $('#products')
                    .accordion("activate", iElem
                    .closest("div[id^=row]")
                    .prev())
                iElem.val(0).select();
        })
    } else if ($(this).val() != "0") {
        row.children().eq(1).text($(this).val())
    } else {
        removeTableRow(row)
    }
})
```

Эта функция-обработчик решает множество задач. Прежде всего, когда пользователь изменяет какое-либо значение, осуществляется проверка того, имеется ли уже в таблице строка, соответствующая данному продукту. В случае отсутствия такой строки создается новая строка, для генерации которой используется шаблон.

```
<script id="rowTmpl" type="text/x-jquery-tmpl">
    <tr id=${name}><td>${product}</td><td>${val}</td>
        <td><a href=#>Remove</a></td></tr>
</script>
```

Необходимые для данного шаблона значения получаются с помощью методов ядра jQuery, которые извлекают информацию из элемента `input`, породившего событие. Мы хотим отображать также названия продуктов, и с этой целью выполняем поиск в DOM-дереве для нахождения ближайшего элемента `label` и считывания его содержимого.

```
$(this).siblings("label").text()
```

Вновь созданная строка присоединяется к таблице. Ряд-заполнитель был скрыт еще раньше в самом начале выполнения функции-обработчика.

```
$('#placeholder').hide();
```

Процесс добавления новых строк таблицы представлен на рис. 25.5. Пользователь вводит значение в текстовом поле, и как только это поле теряет фокус ввода, в корзине покупателя появляется новая позиция.

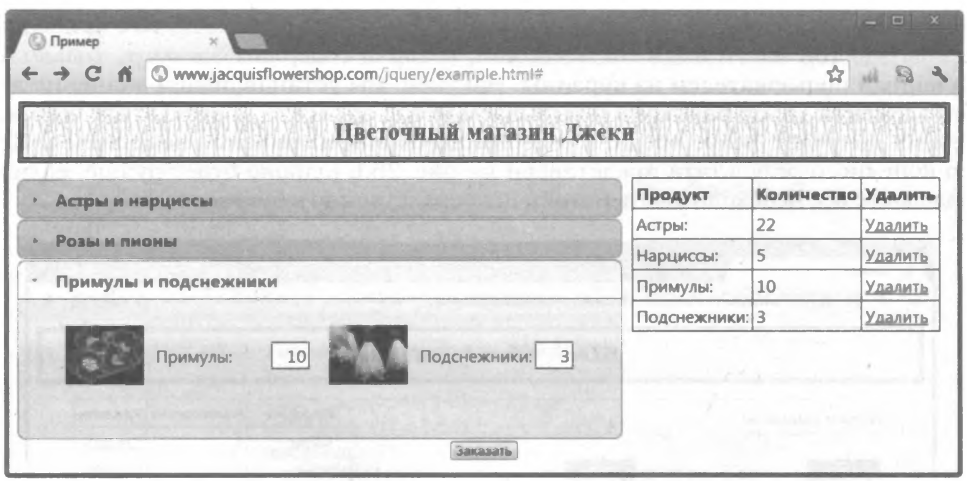


Рис. 25.5. Добавление строк в таблицу корзины

Удаление строк

Вы могли заметить, что в качестве части шаблона данных в документ добавляется элемент `a`. При создании строки по шаблону мы регистрируем функцию-обработчик для этого элемента.

```
...
}).appendTo("#basketTable").find("a").click(function() {
        removeTableRow($(this).closest("tr"));
        var iElem = $('#products')
            .find("input[name=" + fname + "]")
        $('#products')
            .accordion("activate", iElem
            .closest("div[id^=row]")
            .prev())
    iElem.val(0).select();
})
```

Первое, что мы делаем, — вызываем функцию `removeTableRow()`, передавая ей в качестве аргумента элемент `tr`, являющийся ближайшим предком элемента `a`.

Для удаления указанного элемента из документа функция `removeTableRow()` использует метод `remove()`. Она также восстанавливает в таблице ряд-заполнитель в случае отсутствия строк, относящихся к продуктам.

```
function removeTableRow(row) {
    row.remove();
    if ($('#basketTable tbody')
        .children(':visible').length == 1) {
        $('#placeholder').show();
    }
}
```

Удалив строку из таблицы корзины покупателя, мы находим среди продуктов элемент `input`, связанный с данной строкой. Затем мы используем навигацию по DOM-дереву для поиска элемента, являющегося ближайшим предшествующим сестринским элементом по отношению к элементу `div`, который содержит данный элемент `input`, и передаем его методу `activate` виджета Accordion. Это приводит к раскрытию той части виджета Accordion, которая содержит элемент, только что удаленный пользователем из корзины. Наконец, мы устанавливаем значение данного элемента `input` равным `0` и вызываем метод `select()`, в результате чего этот элемент получает фокус ввода, а содержащееся в нем значение выделяется. Пример конечного результата представлен на рис. 25.6 (однако будет лучше, если вы проделаете все необходимые действия непосредственно в браузере).

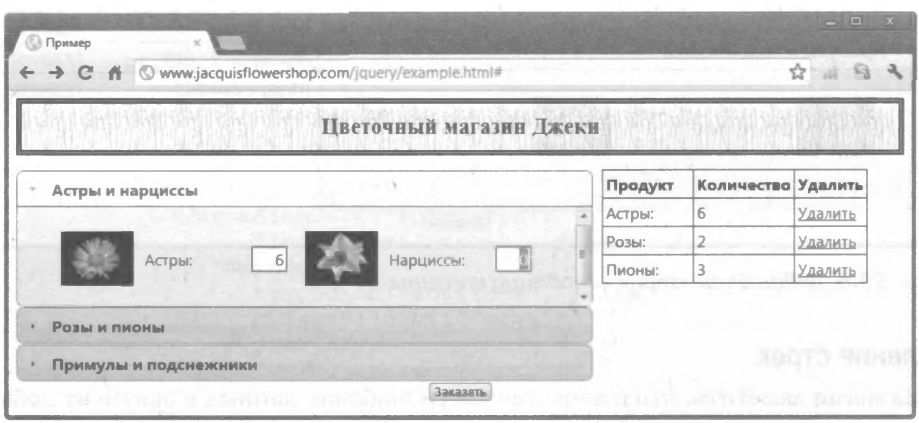


Рис. 25.6. Добавление строк в таблицу корзины

Примечание. Строки удаляются из таблицы и тогда, когда пользователь вводит `0` в текстовом поле, предыдущее значение которого уже представлено строкой в таблице. Это делается с помощью функции `removeTableRow()`, что обеспечивает отображение ряда-заполнителя в необходимых случаях.

Обновление существующих строк

Если в таблице корзины покупателя уже есть строка, соответствующая нужному продукту, то заказываемое количество данного продукта можно изменить. Вместо того чтобы удалять одну строку и создавать другую, мы находим эту строку в таблице и обновляем содержимое нужной ячейки.

```
row.children().eq(1).text($(this).val())
```

Переменная `row` — это объект `jQuery`, содержащий элемент `tr` для продукта в таблице. Доступ к элементу `td` осуществляется по номеру позиции (с помощью метода `eq()`), а его содержимое устанавливается с помощью метода `text()`.

Применение темы оформления

Теперь наша корзина функционирует вполне удовлетворительно, но ее внешний вид оставляет желать лучшего. К счастью, jQuery UI предоставляет библиотеку CSS-стилей (CSS-фреймворк), которые можно применить к элементам, чтобы они выглядели так же, как и виджеты после применения к ним выбранной вами темы стилевого оформления. В листинге 25.4 показано, насколько просто получить требуемый результат путем добавления классов в HTML-элементы документа.

Листинг 25.4. Применение стилей из библиотеки CSS-стилей jQuery UI к элементу `table`

```
...
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="productWrapper">
            <div id="products"></div>
        </div>
        <div id="basket" class="ui-widget ui-widget-content">
            <table border=0 id="basketTable">
                <tr class="ui-widget-header">
                    <th>Продукт</th><th>Количество</th>
                    <th>Удалить</th></tr>
                <tr id="placeholder"><td colspan=3>Продукты не
                    выбраны</td></tr>
            </table>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
...
```

Возможно, вы заметили, что я уже использовал класс `ui-widget` в некоторых примерах в предыдущих главах. Это базовый стиль jQuery UI, и он назначается внешнему контейнеру, содержащему наборы элементов, внешний вид которых должен согласовываться с внешним видом виджетов jQuery UI. Класс `ui-widget-content` назначается элементам, имеющим содержимое, а класс `ui-widget-header`, как несложно догадаться по его названию, используется для заголовков элементов.

Совет. Подробно классы CSS-фреймворка jQuery UI описаны в главе 34.

Для элемента `table` дополнительно к использованию указанных классов задано отсутствие границ (рамки).

```
#basketTable {border: none; border-collapse: collapse}
```

Полученный результат проиллюстрирован на рис. 25.7.

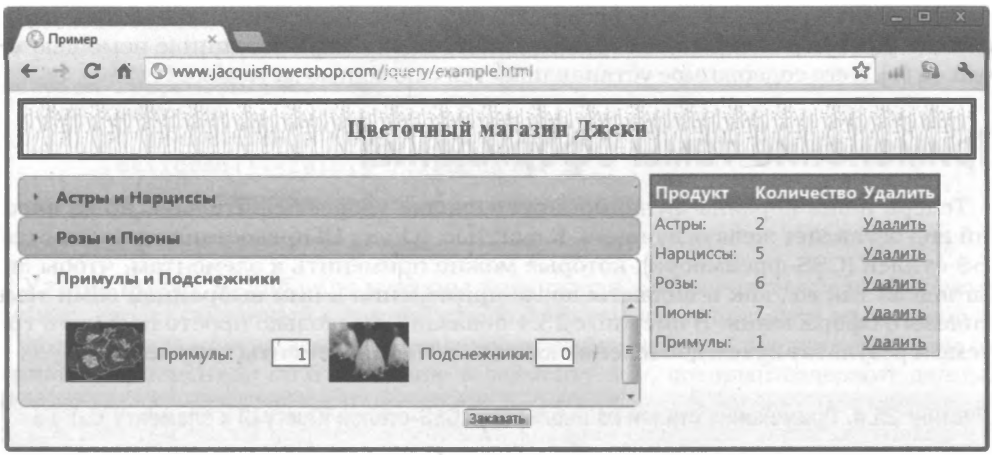


Рис. 25.7. Результат применения классов CSS-фреймворка jQuery UI к таблице

Расширение сферы использования классов CSS-фреймворка

Сферу использования стилей фреймворка в нашем примере можно расширить. Некоторые полезные дополнения к образцу документа приведены в листинге 25.5.

Листинг 25.5. Дополнительное использование стилей CSS-фреймворка

```
...
<body>
    <div id="logoWrapper" class="ui-widget ui-widget-content
        ui-corner-all">
        <h1 id="logo">Цветочный магазин Джеки</h1>
    </div>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="productWrapper">
            <div id="products"></div>
        </div>
        <div id="basket" class="ui-widget ui-widget-content">
            <table border=0 id="basketTable">
                <tr class="ui-widget-header">
                    <th>Продукт</th><th>Количество</th>
                    <th>Удалить</th></tr>
                <tr id="placeholder"><td colspan=3>Продукты не
                    выбраны</td></tr>
            </table>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
...
```

Здесь мы поместили элемент h1 внутрь элемента div и использовали несколько стилей фреймворка, в том числе стиль ui-corner-all, создающий скругленные углы, которые показаны на рис. 25.8. Кроме того, для создания требуемых эффектов в документе применены также некоторые новые стили, заменяющие стили, которые мы используем, начиная с главы 3.

```
...
<style type="text/css">
    .dcell img {height: 60px}
    #basketTable {border: thin solid black;
        border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {width: 30%; text-align: left; float: left;
        margin-left: 10px}
    #buttonDiv {clear: both}
    #logo {font-size: 1.5em; background-size: contain;
        margin: 1px; border: none; color: inherit}
    #logoWrapper {margin-bottom: 5px}
</style>
...
```

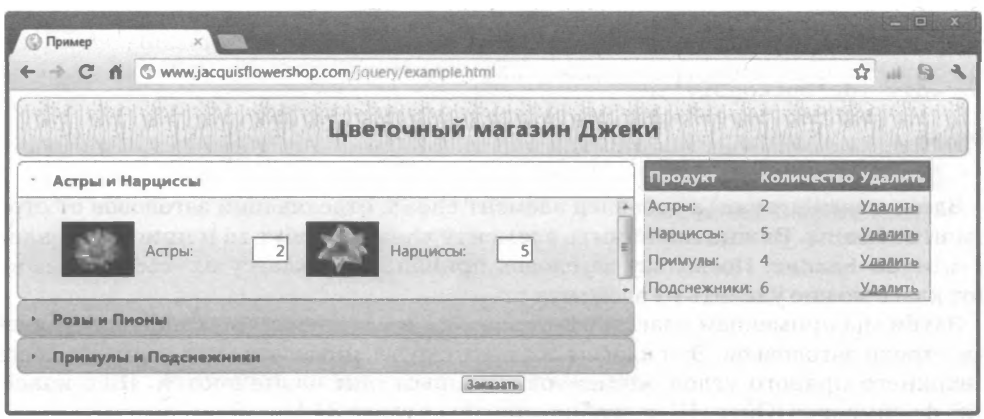


Рис. 25.8. Применение стилей CSS-фреймворка к заголовку документа

Применение скругленных углов в таблице

Применение класса ui-corner-all к элементам table порождает определенные проблемы, как показано на рис. 25.9. Если внимательно присмотреться, то можно заметить, что углы таблицы не скруглены. Это вызвано особенностями взаимодействия классов CSS-фреймворка jQuery UI с теми средствами обработки таблиц, которые используются в большинстве браузеров.

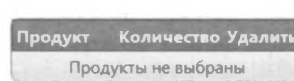


Рис. 25.9. Эффект скругленных углов, примененный к таблице

Для разрешения этой проблемы необходимо изменить элемент `table`, немного иначе применить классы CSS-фреймворка jQuery UI и определить собственный новый стиль. Прежде всего, требуется видоизменить элемент `table`, как показано в листинге 25.6.

Листинг 25.6. Изменение элемента `table` для поддержки скругленных углов

```
...
<form method="post"
    action="http://node.jacquisflowershop.com:9999/order">
    <div id="productWrapper">
        <div id="products"></div>
    </div>
    <div id="basket" class="ui-widget ui-widget-content
        ui-corner-all">
        <table border=0 id="basketTable">
            <thead id="theader" class="ui-widget-header">
                <tr>
                    <th class="ui-corner-tl">Продукт</th>
                    <th>Количество</th>
                    <th class="ui-corner-tr">Удалить</th></tr>
            </thead>
            <tr id="placeholder"><td colspan=3>
                Продукты не выбраны</td></tr>
        </table>
    </div>
    <div id="buttonDiv">
        <button type="submit">Заказать</button></div>
</form>
...
```

Здесь в элемент `table` добавлен элемент `thead`, отделяющий заголовок от строк самой таблицы. Важно назначить элементу `thead` атрибут `id` и присвоить класс `ui-widget-header`. Поскольку заголовок принадлежит классу `ui-widget-header`, этот класс можно удалить из элемента `tr`.

Затем мы применяем классы `ui-corner-tl` и `ui-corner-tr` к наружным ячейкам строки заголовков. Эти классы создают скругленные углы для верхнего левого и верхнего правого углов элементов, которым они назначаются. (Все классы CSS-фреймворка jQuery UI подробно описаны в главе 34.)

Присвоенный элементу `thead` атрибут `id` мы используем для отключения CSS-свойства `border` в элементе `style`, и делаем то же самое в отношении элемента `table`.

```
...
<style type="text/css">
    .dcell img {height: 60px}
    #basketTable {border: none; border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {width: 30%; text-align: left; float: left;
        margin-left: 10px}
    #buttonDiv {clear: both}
    #logo {font-size: 1.5em; background-size: contain;
        margin: 1px; border: none; color: inherit}
    #logoWrapper {margin-bottom: 5px}
```

```
    #theader {border: none}
</style>
...
```

Наконец, нам придется внести небольшое исправление в функцию `removeTableRow()`. В результате отделения строки заголовков и помещения ее в элемент `thead` количество строк в элементе `tbody` уменьшается на единицу. Вот это изменение.

```
function removeTableRow(row) {
    row.remove();
    if ($('#basketTable tbody')
        .children(':visible').length == 0) {
        $('#placeholder').show();
    }
}
```

• **Совет.** Элемент `tbody` автоматически создается браузером при синтаксическом анализе элемента `table`. Особенностью HTML является то, что вы не обязаны определять этот элемент (хотя и можете это сделать, если захотите).

После внесения перечисленных изменений вы получите таблицу с скругленными углами, которая согласуется с другими элементами документа, как показано на рис. 25.10.

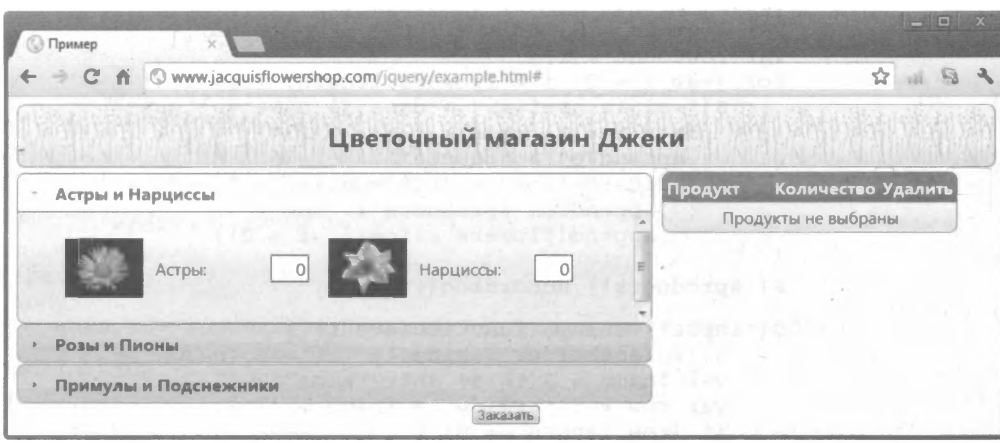


Рис. 25.10. Таблица со скругленными углами

Создание кнопки jQuery UI

Наш следующий шаг — перемещение кнопки в другое место и ее преобразование в виджет jQuery UI. Изменения, которые для этого требуется внести, указаны в листинге 25.7.

Листинг 25.7. Перемещение и преобразование кнопки

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
```

```
<script src="jquery-1.7.js" type="text/javascript"></script>
<script src="jquery-ui-1.8.16.custom.js"
    type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<link rel="stylesheet" type="text/css"
    href="jquery-ui-1.8.16.custom.css"/>
<script src="jquery.tmpl.js" type="text/javascript"></script>
<style type="text/css">
    .dcell img {height: 60px}
    #basketTable {border: none; border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {text-align: left;}
    #buttonDiv {clear: both; margin: 5px}
    #logo {font-size: 1.5em; background-size: contain;
        margin: 1px; border: none; color: inherit}
    #logoWrapper {margin-bottom: 5px}
    #theader {border: none}

</style>
<script type="text/javascript">
    $(document).ready(function() {
        $.getJSON("mydata.json", function(data) {
            var flowers = $('#flowerTmpl').tmpl(data);
            var rowCount = 1;
            for (var i = 0; i < flowers.length; i += 2) {
                $("<h2><a href=#>" + data[i].name + " и " +
                  data[i + 1].name + "</a></h2>")
                  .appendTo("#products");
                $("<div id='row" + (rowCount++) + "'></div>")
                    .appendTo("#products")
                    .append(flowers.slice(i, i + 2))
            }
            $('#products').accordion();

            $('input').change(function(event) {
                $('#placeholder').hide();
                var fname = $(this).attr("name");
                var row = $('tr[id=' + fname + ']');
                if (row.length == 0) {
                    $('#rowTmpl').tmpl({
                        name: fname,
                        val:  $(this).val(),
                        product: $(this).siblings("label")
                            .text()
                    }).appendTo("#basketTable")
                      .find("a").click(function() {
                        removeTableRow($(this).closest("tr"));
                        var iElem = $('#products')
                            .find("input[name=" + fname + "]")
                        $('#products').accordion("activate",
                            iElem.closest("div[id^=row]")
                                .prev())
                        iElem.val(0).select();
                    })
                } else if ($(this).val() != "0") {
```

```
                    row.children().eq(1).text($(this).val())
                } else {
                    removeTableRow(row)
                }
            })
        });

        $('#buttonDiv, #basket').wrapAll("<div />")
            .parent().css({
                float: "left",
               marginLeft: "2px"
         })

         $('button').button()

        function removeTableRow(row) {
            row.remove();
            if ($('#basketTable tbody')
                .children(':visible').length == 0) {
                    $('#placeholder').show();
            }
        }
    });
</script>
<script id="rowTmpl" type="text/x-jquery-tmpl">
    <tr id=${name}><td>${product}</td><td>${val}</td>
        <td><a href=#>Удалить</a></td></tr>
</script>
<script id="flowerTmpl" type="text/x-jquery-tmpl">
    <div class="dcell">
        <img src="${product}.png"/>
        <label for="${product}">${name}:</label>
        <input name="${product}" value="0" />
    </div>
</script>
</head>
<body>
    <div id="logoWrapper" class="ui-widget ui-widget-content
        ui-corner-all">
        <h1 id="logo">Цветочный магазин Джеки</h1>
    </div>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="productWrapper">
            <div id="products"></div>
        </div>
        <div id="basket" class="ui-widget ui-widget-content
            ui-corner-all">
            <table border=0 id="basketTable">
                <thead id="theader" class="ui-widget-header">
                    <tr>
                        <th class="ui-corner-tl">Продукт</th>
                        <th>Количество</th>
                        <th class="ui-corner-tr">Удалить</th></tr>
                </thead>
                <tr id="placeholder"><td colspan=3>
                    Продукты не выбраны</td></tr>
            </table>
```

```
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
</body>
</html>
```

Здесь мы поместили элементы `buttonDiv` и `basket` в новый элемент `div` и изменили некоторые CSS-стили для настройки позиций этих элементов. И наконец, мы вызываем метод `button()` для создания кнопки jQuery UI, как показано на рис. 25.11.

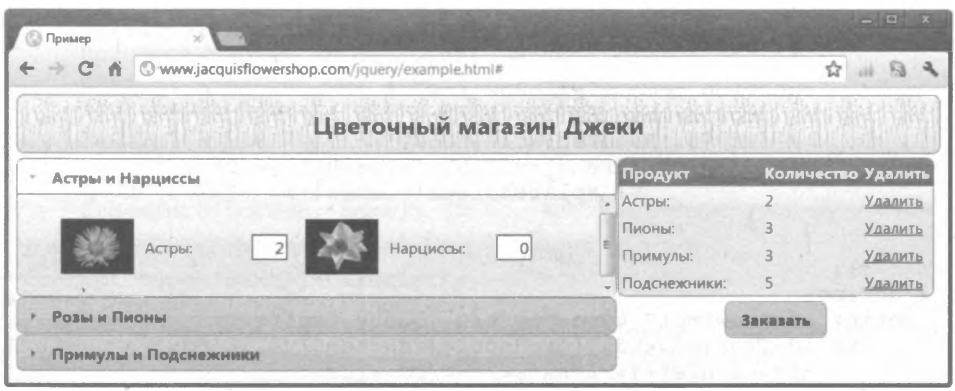


Рис. 25.11. Перемещение и преобразование элемента `button`

Добавление диалогового окна для завершения заказа

Мы хотим, чтобы, прежде чем завершить оформление заказа, щелкнув на кнопке Заказать, пользователь предоставил нам некоторую дополнительную информацию о себе. В главе 20 уже было показано, каким образом можно использовать вкладки для отображения форм, состоящих из нескольких частей. На этот раз, чтобы внести некоторое разнообразие, воспользуемся виджетом Dialog. Изменения, которые для этого потребуется внести в документ, представлены в листинге 25.8.

Листинг 25.8. Добавление виджета Dialog

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <script src="jquery.tmpl.js" type="text/javascript"></script>
    <style type="text/css">
        .dcell img {height: 60px}
```

```
    #basketTable {border: none; border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {text-align: left;}
    #buttonDiv {clear: both; margin: 5px}
    #logo {font-size: 1.5em; background-size: contain;
        margin: 1px; border: none; color: inherit}
    #logoWrapper {margin-bottom: 5px}
    #theader {border: none}
    #completeDialog input {width: 150px; margin-left: 5px;
        text-align: left}
    #completeDialog label {width: 60px; text-align: right}
</style>
<script type="text/javascript">
    $(document).ready(function() {
        $.getJSON("mydata.json", function(data) {
            var flowers = $('#flowerTmpl').tmpl(data);
            var rowCount = 1;
            for (var i = 0; i < flowers.length; i += 2) {
                $("<h2><a href=#>" + data[i].name + " и " +
                  data[i + 1].name + "</a></h2>")
                  .appendTo("#products");
                $("<div id='row" + (rowCount++) + "'></div>")
                    .appendTo("#products")
                    .append(flowers.slice(i, i + 2))
            }
            $('#products').accordion();

            $('#products input').change(function(event) {
                $('#placeholder').hide();
                var fname = $(this).attr("name");
                var row = $('tr[id=' + fname + ']');
                if (row.length == 0) {
                    $('#rowTmpl').tmpl({
                        name: fname,
                        val:  $(this).val(),
                        product: $(this).siblings("label")
                            .text()
                    }).appendTo("#basketTable")
                       .find("a").click(function() {
                        removeTableRow($(this).closest("tr"));
                        var iElem = $('#products')
                            .find("input[name=" + fname + "]")
                        $('#products').accordion("activate",
                            iElem.closest("div[id^=row]")
                                .prev())
                        iElem.val(0).select();
                    })
                } else if ($(this).val() != "0") {
                    row.children().eq(1).text($(this).val())
                } else {
                    removeTableRow(row)
                }
            })
        });
```

```
            $('#buttonDiv, #basket').wrapAll("<div />")
                .parent().css({
                    float: "left",
                   marginLeft: "2px"
                })

                $('button').button()

                $('#completeDialog').dialog({
                    modal: true,
                    buttons: [{text: "OK", click: sendOrder},
                              {text: "Отменить", click: function() {
                                $("#completeDialog").dialog("close");
                    }}]
                });

                function sendOrder() {

                }

                function removeTableRow(row) {
                    row.remove();
                    if ($('#basketTable tbody')
                        .children(':visible').length == 0) {
                            $('#placeholder').show();
                    }
                }
        });
    </script>
    <script id="rowTmpl" type="text/x-jquery-tmpl">
        <tr id=${name}><td>${product}</td><td>${val}</td>
            <td><a href=#>Удалить</a></td></tr>
    </script>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
        <div class="dcell">
            <img src="${product}.png"/>
            <label for="${product}">${name}:</label>
            <input name="${product}" value="0" />
        </div>
    </script>
</head>
<body>
    <div id="logoWrapper" class="ui-widget ui-widget-content
        ui-corner-all">
        <h1 id="logo">Цветочный магазин Джеки</h1>
    </div>
    <form method="post"
        action="http://node.jacquisflowershop.com:9999/order">
        <div id="productWrapper">
            <div id="products"></div>
        </div>
        <div id="basket" class="ui-widget ui-widget-content
            ui-corner-all">
            <table border=0 id="basketTable">
                <thead id="theader" class="ui-widget-header">
                    <tr>
                        <th class="ui-corner-tl">Продукт</th>
                        <th>Количество</th>
                        <th class="ui-corner-tr">Удалить</th></tr>
```

```
                </thead>
                <tr id="placeholder"><td colspan=3>
                    Продукты не выбраны</td></tr>
            </table>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
    <div id="completeDialog" title="Завершите покупку">
            <div><label for="name">Имя: </label>
                <input name="first" /></div>
            <div><label for="email">Email: </label>
                <input name="email" /></div>
            <div><label for="city">Город: </label>
                <input name="city" /></div>
    </div>
</body>
</html>
```

Здесь мы добавили элемент `div`, содержимое которого будет отображаться для пользователя в элементе `body`, а также некоторые CSS-стили, заменяющие стили из файла `styles.css`, которые импортируются в документ с помощью элемента `link`. Для создания диалогового окна используется следующий вызов метода `dialog()`.

```
$('#completeDialog').dialog({
    modal: true,
    buttons: [{text: "OK", click: sendOrder},
        {text: "Cancel", click: function() {
        $("#completeDialog").dialog("close");
    }}]
});
```

Здесь мы создаем модальный вариант диалогового окна, в котором есть две кнопки. После щелчка на кнопке Отменить диалоговое окно закрывается. Щелчок на кнопке OK приводит к вызову функции `sendOrder()`. Пока что эта функция ровным счетом ничего не делает.

По умолчанию диалоговое окно открывается сразу же после его создания (см. главу 22) и выглядит так, как на рис. 25.12.

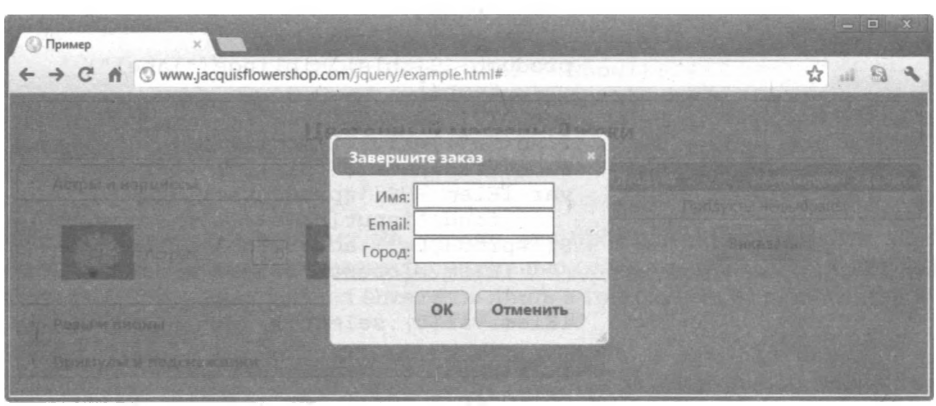


Рис. 25.12. Диалоговое окно, используемое для завершения процедуры покупки

Совет. Обратите внимание на сужение набора выбираемых элементов `input` при установке для них обработчика событий `change`. Это делается для того, чтобы предотвратить привязку функции-обработчика к элементам `input` диалогового окна. Если этого не сделать, то ввод значений в диалоговом окне будет приводить к появлению в корзине новых записей.

Обработка щелчка на кнопке Заказать

Мы должны сделать так, чтобы диалоговое окно появлялось лишь после того, как пользователь щелкнет на кнопке Заказать. Для сокрытия диалогового окна до тех пор, пока в нем не возникнет необходимость, мы используем параметр `autoOpen`, а для обработки щелчка на кнопке — метод `click()`, как показано в листинге 25.9.

Листинг 25.9. Сокрытие диалогового окна и обработка щелчка на кнопке

```
<script type="text/javascript">
    $(document).ready(function() {
        $.getJSON("mydata.json", function(data) {
            var flowers = $('#flowerTmpl').tmpl(data);
            var rowCount = 1;
            for (var i = 0; i < flowers.length; i += 2) {
                $("<h2><a href=#>" + data[i].name + " и " +
                  data[i + 1].name.toLowerCase() +
                      "</a></h2>").appendTo("#products");
                $("<div id='row" + (rowCount++) + "'></div>")
                    .appendTo("#products")
                    .append(flowers.slice(i, i + 2))
            }
            $('#products').accordion();

            $('#products input').change(function(event) {
                $('#placeholder').hide();
                var fname = $(this).attr("name");
                var row = $('tr[id=' + fname + ']');
                if (row.length == 0) {
                    $('#rowTmpl').tmpl({
                        name: fname,
                        val:  $(this).val(),
                        product: $(this).siblings("label")
                            .text()
                    }).appendTo("#basketTable")
                      .find("a").click(function() {
                        removeTableRow($(this).closest("tr"));
                        var iElem = $('#products')
                            .find("input[name=" + fname + "]")
                        $('#products').accordion("activate",
                            iElem.closest("div[id^=row]")
                                .prev())
                        iElem.val(0).select();
                    })
                } else if ($(this).val() != "0") {
                    row.children().eq(1).text($(this).val())
                } else {
                    removeTableRow(row)
```

```
                }
            })
        });

        $('#buttonDiv, #basket').wrapAll("<div />")
            .parent().css({
                float: "left",
               marginLeft: "2px"
         })

         $('button').button().click(function(e) {
            e.preventDefault();
            if ($('#placeholder:visible').length) {

                $('<div>Пожалуйста, выберите продукт</div>')
                  .dialog({
                      modal: true,
                      buttons: [{text: "OK",
                          click: function() {$(this)
                          .dialog("close")}}]
                  })
            } else {
                $('#completeDialog').dialog("open");
            }
         })

         $('#completeDialog').dialog({
            modal: true,
            autoOpen: false,
            buttons: [{text: "OK", click: sendOrder},
                      {text: "Отменить", click: function() {
                        $("#completeDialog").dialog("close");
            }}]
        });

        function sendOrder() {

        }

        function removeTableRow(row) {
            row.remove();
            if ($('#basketTable tbody')
                .children(':visible').length == 0) {
                    $('#placeholder').show();
            }
        }
    });
</script>
...
```

Когда пользователь щелкает на кнопке, мы проверяем, является ли элемент `placeholder` видимым. Это делается с помощью селектора jQuery, предоставляющего объект, который содержит элементы лишь в том случае, если указанный элемент виден на экране.

Здесь видимость элемента `placeholder` используется для индикации того, выбран ли пользователем хотя бы один продукт. Если в корзине покупателя есть хотя бы один продукт, этот элемент скрыт, и его появление говорит о том, что ни один продукт выбран не был.

Совет. Описанный прием служит хорошим примером реализации функциональности приложения на нескольких уровнях. Вместе с тем использованный подход к проверке того, выбрал ли пользователь продукты, зависит от способа реализации корзины покупателя, и если этот способ впоследствии будет изменен, то процедуру проверки также потребуется изменить.

Если пользователь выполняет щелчок на кнопке, не выбрав ни одного продукта, динамически создается и отображается диалоговое окно, представленное на рис. 25.13. В случае выбора пользователем продуктов открывается диалоговое окно, в котором накапливается информация о количестве выбранных пользователем продуктов.

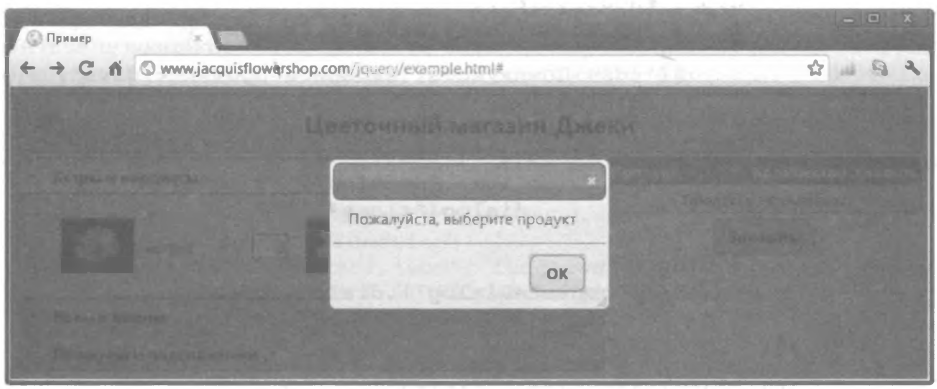


Рис. 25.13. Диалоговое окно, напоминающее о необходимости выбрать продукт

Завершение оформления заказа

Теперь нам осталось лишь реализовать функцию `sendOrder()`. Ранее я уже продемонстрировал несколько возможных способов отправки данных на сервер посредством Ajax, и поэтому в данном примере, чтобы не усложнять его, я просто получаю необходимые значения из различных текстовых полей ввода и создаю объект JSON, который может быть отправлен на сервер для последующей обработки. Соответствующие исправления, которые требуется внести в документ, представлены в листинге 25.10.

Листинг 25.10. Завершение процедуры заказа

```
...
function sendOrder() {
    var data = new Object();
    $('input').each(function(index, elem) {
        var jqElem = $(elem);
        data[jqElem.attr("name")] = jqElem.val();
    })
    console.log(JSON.stringify(data));
    $('#completeDialog').dialog("close");
    $('#products input').val("0");
    $('#products').accordion("activate", 0)
    $('#basketTable tbody').children(':visible').remove();
    $('#placeholder').show();
}
...
```

В этой функции мы получаем значения, содержащиеся в каждом из текстовых полей, и добавляем их в виде свойств в объект, который затем преобразуем в формат JSON и выводим на консоль.

Далее мы возвращаем документ в исходное состояние путем закрытия диалогового окна, сброса значений в текстовых полях, перехода на первую вкладку виджета Accordion и очистки корзины. Документ, в котором был сделан выбор некоторых продуктов, представлен на рис. 25.14. Он будет использован для генерации строки JSON.

После щелчка на кнопке Заказать открывается диалоговое окно, предлагающее вести дополнительную информацию, как показано на рис. 25.15.

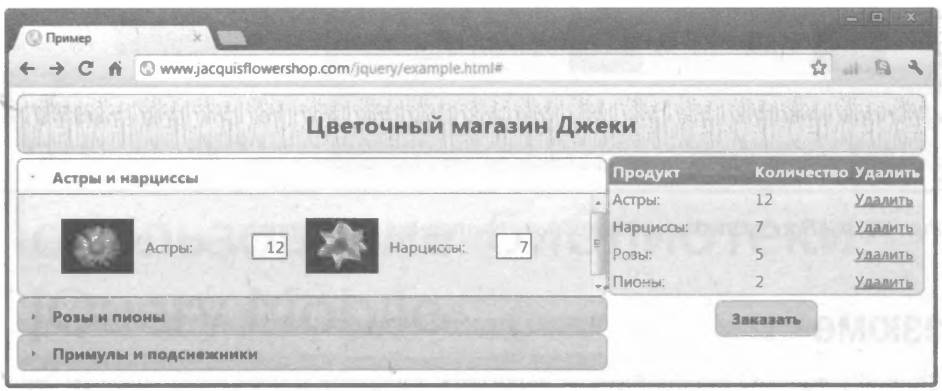


Рис. 25.14. Выбор продуктов с использованием примера документа

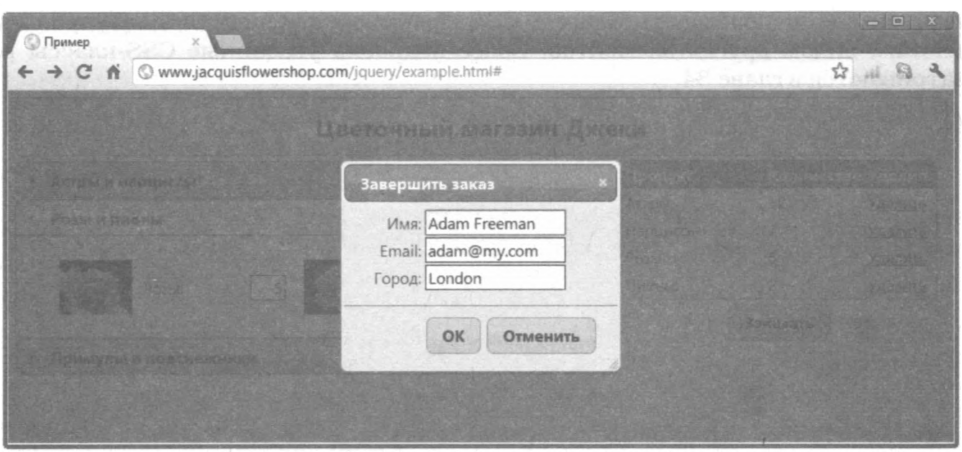


Рис. 25.15. Предоставление дополнительной информации для завершения оформления заказа

Наконец, щелчок на кнопке OK приводит к генерации данных в формате JSON и восстановлению исходного состояния документа. Консольный вывод имеет следующий вид.

```
{"astor":"12","daffodil":"7","rose":"5","peony":"2",
    "primula":"0","snowdrop":"0", "first":"Adam Freeman",
    "email":"adam@my.com","city":"London"}
```

Документ, возвращенный в исходное состояние, в котором он готов к приему следующих заказов, представлен на рис. 25.16.

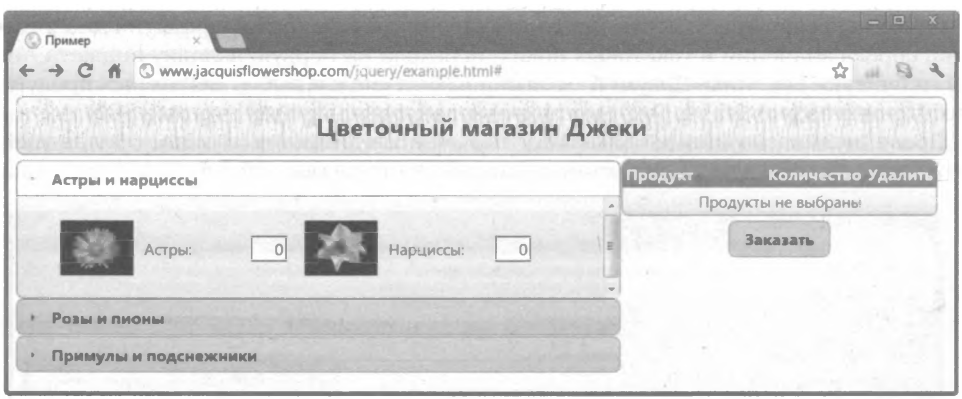


Рис. 25.16. Восстановление исходного состояния документа

Резюме

В этой главе мы переработали документ, включив в него возможности, предлагаемые библиотекой jQuery UI. В него были добавлены виджеты Accordion, Dialog и Button, и вы имели возможность получить первое представление о том, каким образом можно использовать классы CSS-фреймворка jQuery UI для управления внешним видом других элементов. Более подробно указанные CSS-классы рассматриваются в главе 34.

Часть V

Использование библиотеки jQuery Mobile

ГЛАВА 26

Знакомство с jQuery Mobile

В этой главе описываются процедуры загрузки библиотеки jQuery Mobile и ее подключения к HTML-документу, анализируются отличия принятого в jQuery Mobile подхода к созданию виджетов от того подхода, который был рассмотрен в предыдущей части, и рассматриваются особенности разработки веб-приложений для мобильных устройств. Сенсорные устройства ставят перед разработчиками веб-приложений сложные задачи, и поэтому очень важно знать некоторые базовые функции библиотеки jQuery Mobile, предназначенные для упрощения разработки, и общие рекомендации по разработке и тестированию мобильных веб-приложений. Перечень тем, рассматриваемых в данной главе, приведен в табл. 26.1.

Таблица 26.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Добавление функциональности jQuery Mobile в HTML-документ	Добавьте в документ элемент `script` для импорта библиотек jQuery и jQuery Mobile и элемент `link` для импорта CSS-файлов	1
Создание страницы jQuery Mobile	Используйте атрибут `data-role` со значением `page`	2
Отключение виртуальной страницы браузера	Настройте окно просмотра	3
Отсрочка выполнения пользовательского JavaScript-кода до тех пор, пока документ не будет улучшен средствами jQuery Mobile	Используйте событие `pageinit`	4
Упрощение обработки сенсорных событий	Используйте жесты и виртуальные события мыши	5–7
Реагирование на изменение ориентации устройства	Организуйте обработку события `orientationchange` или используйте медиазапросы CSS	8, 9

Подготовка библиотеки jQuery Mobile к работе

Эта глава начинается с рассмотрения процесса загрузки и установки библиотеки jQuery Mobile. Библиотека jQuery Mobile реализована поверх библиотек jQuery и jQuery UI, и поэтому не удивительно, что она имеет сходную с ними процедуру установки.

Загрузка jQuery Mobile

Прежде всего, вам понадобится библиотека jQuery Mobile, которая доступна для загрузки по адресу `http://jquerymobile.com/`. На момент написания книги текущей версией jQuery Mobile была 1.0, но даже в случае выхода новых версий вы сможете получить Zip-файл библиотеки на странице Download сайта jQuery Mobile по указанному адресу. Версии 1.0 соответствует файл `mobile-1.0.zip`.

Совет. Как и в случае библиотек jQuery и jQuery UI, для загрузки библиотеки jQuery Mobile можно использовать сети доставки контента (CDN). О сетях CDN уже говорилось в главе 5, и эта возможность отлично подходит для веб-приложений, развертываемых в Интернете (чего, как правило, нельзя сказать в отношении приложений для интрасетей). На странице загрузки jQuery Mobile приведен подробный перечень ссылок, которые потребуются для удаленного использования jQuery Mobile, обеспечиваемого сетями CDN.

Создание темы

Библиотека jQuery Mobile поддерживает фреймворк тем оформления, фактически являющийся упрощенной версией аналогичного фреймворка jQuery UI. Пакет jQuery Mobile включает стандартную тему, но можно также создать собственную тему, посетив сайт `http://jquerymobile.com/themoroller`. Настройщик тем Themoroller сгенерирует согласно вашим пожеланиям Zip-файл, содержащий CSS-стили, которые вы сможете включать в свои документы. Об использовании данного фреймворка подробно говорится в главе 28, но в приводимых в книге примерах используется стандартная тема, установленная по умолчанию. Дело в том, что первоначальная версия фреймворка работает не совсем стабильно, хотя можно надеяться, что к тому времени, когда вы будете читать эти строки, проблема уже будет устранена.

Загрузка jQuery

Для работы с jQuery Mobile также нужна библиотека jQuery. Следует обратить внимание на то, что библиотека jQuery Mobile 1.0 предназначена для работы только с библиотекой jQuery версии 1.6.4. Такая ситуация не должна продолжаться долго, и я надеюсь, что к тому времени, когда вы будете читать эти строки, проблема потеряет свою актуальность. Для получения старой версии библиотеки jQuery перейдите по ссылке `http://code.jquery.com/index.htm` и выберите пункт All jQuery Versions в меню страницы.

Все версии доступны в двух форматах — полном (несжатом) и сокращенном (сжатом). Для разработки проектов лучше подходит полная версия, поскольку это позволяет выполнять отладку кода вместе с кодом jQuery. Имена файлов сокращенных версий (текст которых не структурирован и не содержит комментариев, благодаря чему занимает меньше места и быстрее загружается) содержат суффикс `min`. Для загрузки полной версии выберите файл `jquery-1.6.4.js`.

Совет. Несмотря на то что библиотека jQuery Mobile реализована поверх библиотеки jQuery UI, устанавливать jQuery UI необязательно. Все, что необходимо для нормальной работы библиотеки jQuery Mobile, содержится в ее загрузочном файле.

Установка jQuery Mobile

Для установки библиотеки jQuery Mobile нужно скопировать на свой сайт несколько файлов. Один из них — это файл `jquery-1.6.4.js`, который можно загрузить с сайта `http://jquery.com`. Остальные необходимые файлы и папки находятся в загрузочном архиве библиотеки jQuery Mobile:

- `jquery.mobile-1.0.js` (JavaScript-библиотека jQuery Mobile);
- `jquery.mobile-1.0.css` (CSS-стили, используемые библиотекой jQuery Mobile);
- папка `images` (значки, используемые библиотекой jQuery Mobile).

Скопировав эти файлы, можно создать документ, использующий функциональность jQuery Mobile. Мой файл примера называется `example.html` и находится в той же папке, что и перечисленные выше файлы. Содержимое файла примера приведено в листинге 26.1.

Листинг 26.1. Содержимое файла `example.com`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet" href="jquery.mobile-1.0.css"
        type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="header">
                <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><button>Нажми меня</button></p>
        </div>
   </div>
</body>
</html>
```

Полужирным шрифтом в документе выделены элементы, необходимые для jQuery Mobile. Элементы `script` импортируют JavaScript-библиотеки jQuery и jQuery Mobile, а элемент `link` — CSS-стили, на использовании которых основана работа jQuery Mobile. Поскольку мой HTML-файл находится в одной папке с файлами JavaScript и CSS, для ссылок на них мне было достаточно указывать только их имена, а не полный URL-адрес.

Совет. Не обращайте пока внимания на остальную часть документа. Вскоре я объясню, для чего предназначен элемент `meta` и как используется содержимое элемента `body`.

Особенности подхода, используемого в jQuery Mobile

Несмотря на то что библиотека jQuery Mobile реализована поверх библиотеки jQuery UI, между ними имеются некоторые существенные различия, о которых следует знать. Прежде чем переходить к более подробному обсуждению возможностей библиотеки jQuery Mobile, имеет смысл рассмотреть эти различия, чтобы вам было понятно все, о чем будет идти речь в последующих главах.

Многоуровневая поддержка

Библиотека jQuery Mobile предлагает разные уровни поддержки для разных типов мобильных браузеров. В настоящее время предусмотрено три класса поддержки, каждому из которых соответствует обширный перечень поддерживаемых устройств и браузеров. Класс A, которому соответствует максимальный уровень поддержки, обеспечивает наиболее комфортные условия работы для пользователей и реализует все функциональные возможности, описанные в данной части.

Классу B соответствуют практически те же возможности, что и классу A, за исключением навигации с помощью Ajax, которая описана в главе 27. Этот класс обеспечивает вполне приемлемую функциональность, но переходы между страницами приложения выполняются не так плавно, как в случае устройств класса A.

Классу C соответствует простейший сервис. К этой категории относятся устаревшие устройства, которым jQuery Mobile может добавить лишь незначительную дополнительную функциональность.

К счастью, большинство современных мобильных устройств относится к категории, которой соответствует поддержка класса A. С подробным списком поддерживаемых устройств можно ознакомиться по следующему адресу:

```
http://jquerymobile.com/demos/1.0/docs/about/platforms.html
```

Автоматическое улучшение

Самым заметным отличием, с которым вы сталкиваетесь в процессе использования jQuery Mobile, является то, что не нужно заботиться о создании виджетов. При работе с jQuery UI вы выбираете один или несколько элементов с помощью jQuery, а затем применяете к ним конкретный метод, например `button()` или `tabs()`, вызов которого необходим для создания виджета определенного типа. Взглянув на листинг 26.1, вы не заметите в нем ни одного элемента `script`, с помощью которого создавались бы виджеты, а те элементы `script`, которые присутствуют в документе, всего лишь импортируют библиотеки jQuery и jQuery Mobile. Тем не менее вы получаете форматированное содержимое, как показано на рис. 26.1. (Этот рисунок был получен с помощью эмулятора Opera Mobile, который будет интенсивно использоваться в этой части и о котором речь пойдет далее.)

Примечание. Для получения многих рисунков, приведенных в этой части, использовался эмулятор браузера Opera Mobile с разрешением 480×320 пикселей. Такое разрешение, типичное для смартфонов (хотя и не самое высокое, если иметь в виду новейшие модели устройств), позволяет разместить на одной странице больше примеров. Несмотря на то что этого разрешения вполне достаточно для разработки, я рекомендую тестировать реальные продукты в расширенном диапазоне разрешений экрана, которые встречаются чаще всего.

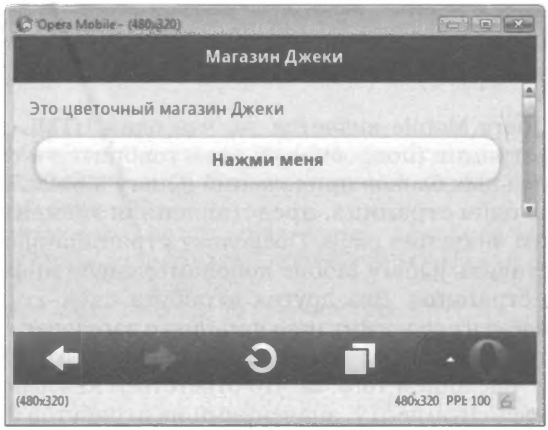


Рис. 26.1. Пример документа

После подключения библиотеки jQuery Mobile к документу с помощью элемента `script` качество страниц немедленно улучшается. Прежде всего jQuery Mobile ищет на странице элементы с атрибутом `data-role`, значения которого позволяют судить о том, какие функции данный элемент выполняет в документе. Основываясь на значениях этих атрибутов, jQuery Mobile определяет, как именно должен отображаться элемент на экране. В листинге 26.2 атрибуты `data-role` элементов выделены полужирным шрифтом.

Совет. Атрибуты, имена которых начинаются с префикса `data`, называются *атрибутами данных* (data attributes). Одно время атрибуты данных являлись частью неофициальных соглашений, которые регламентировали правила определения пользовательских атрибутов, но впоследствии были включены в официальный стандарт HTML5.

Листинг 26.2. Атрибуты `data-role` в примере документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="header">
                <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><button>Нажми меня</button></p>
```

```
        </div>
    </div>
</body>
</html>
```

Особенностью jQuery Mobile является то, что один HTML-документ может содержать несколько страниц (подробнее об этом говорится в главе 27). Страницы играют роль строительных блоков приложений jQuery Mobile. В приведенном примере имеется только одна страница, представленная элементом `div` с атрибутом `data-role`, имеющим значение `page`. Поскольку страница вложена в HTML-документ, нужно предоставить jQuery Mobile дополнительную информацию о роли остальных элементов страницы. Два других атрибута `data-role` сообщают jQuery Mobile о том, какой элемент содержит информацию о заголовке страницы, а какой — ее содержимое. Используемые в нашем примере значения атрибута `data-role` приведены в табл. 26.2 с указанием того, за что ответствен каждый из них. Установить взаимосвязь между элементами `div`, значениями их атрибутов `data-role` и структурой страницы, представленной на рис. 26.1, не составит для вас большого труда.

Совет. Библиотека jQuery Mobile автоматически создает оболочку для той части страницы, которая является ее содержимым. Это означает, что любые элементы, не являющиеся частью другой секции документа, считаются содержимым, что позволяет опускать явное объявление элемента для этой секции.

Таблица 26.2. Значения атрибута `data-role`, используемые в примере документа

Значение	*Описание*
`page`	Сообщает jQuery Mobile, что содержимое данного элемента следует интерпретировать как страницу
`header`	Сообщает jQuery Mobile, что содержимое данного элемента следует интерпретировать как заголовок страницы
`content`	Сообщает jQuery Mobile, что содержимое данного элемента следует интерпретировать как содержимое страницы

Вам не нужно предпринимать никаких действий для того, чтобы инициировать поиск элементов с атрибутами `data-role` и генерацию страницы. Все это происходит автоматически при загрузке HTML-документа. К некоторым элементам, таким как `button`, автоматически применяется стилевое оформление (хотя, как будет показано в последующих главах, внешний вид большинства виджетов можно настраивать с помощью других атрибутов данных).

Совет. Разработчики библиотеки jQuery Mobile многое сделали для того, чтобы свести к минимуму объем пользовательского JavaScript-кода, необходимого для создания мобильного веб-приложения. По сути, простые приложения можно создавать, вообще не написав ни одной строки собственного JavaScript-кода. В то же время не следует воспринимать это утверждение так, будто ничто не мешает создавать приложения jQuery Mobile для браузеров, на которых выполнение сценариев JavaScript запрещено. Библиотека jQuery Mobile — это библиотека JavaScript, и для того, чтобы она могла автоматически улучшать страницы, наличие поддержки JavaScript является обязательным условием.

Окно просмотра

Выделенный в приведенном ниже листинге 26.3 элемент не является частью jQuery Mobile, однако выполняет важные функции, будучи добавленным в HTML-документ.

Листинг 26.3. Настройка окна просмотра с помощью элемента `meta`

```
...
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
...
```

Здесь выделен элемент `meta` с атрибутом `name`, которому присвоено значение `viewport`. Многие мобильные браузеры улучшают совместимость с веб-сайтами, ориентированными на браузеры настольных компьютеров, используя для отображения веб-содержимого *виртуальные страницы* (virtual pages). В целом это здравая идея, поскольку благодаря этому пользователь получает цельное представление о структуре страницы посредством миниатюр, даже если детали слишком мелкие для того, чтобы их можно было прочитать. На рис. 26.2 показана главная страница сайта jQuery Mobile в исходном виде и после увеличения масштаба до значений, обеспечивающих возможность прочтения текста.

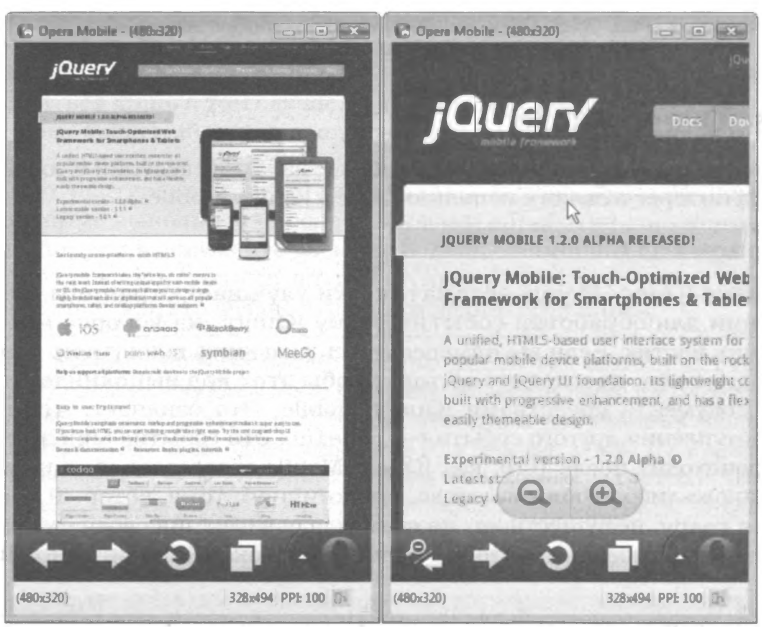


Рис. 26.2. Виртуальная страница мобильного браузера

На первом снимке веб-сайт jQuery Mobile представлен с использованием портретной ориентации страницы (которая усиливает эффект). Размер текста слишком мал, чтобы его можно было прочитать, но в мобильных браузерах предусмотрена поддержка увеличения масштаба отдельных областей страницы, как показано на

втором снимке. Несомненно, виртуальные страницы являются компромиссным решением, но это решение оправдано, если учесть, что число сайтов, приспособленных для работы с мобильными устройствами, относительно невелико.

Дело в том, что виртуальная страница применяется без сколь-нибудь серьезного учета специфики конкретных условий просмотра, тем самым создавая проблемы для приложений jQuery Mobile. На рис. 26.3 показано, как отображается образец документа в случае использования виртуальной страницы.

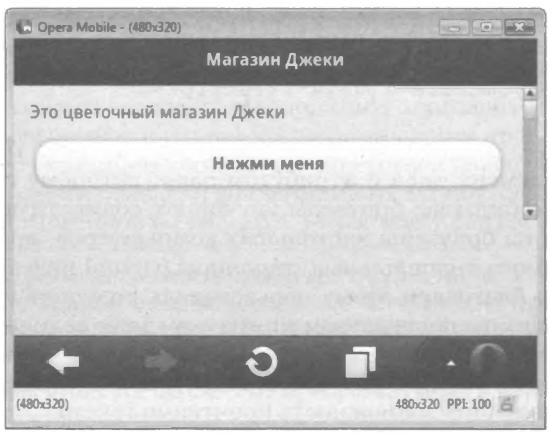


Рис. 26.3. Образец документа, отображаемый на странице с альбомной ориентацией

События jQuery Mobile

В следующих разделах приведены наиболее важные сведения о событиях, представляющих интерес в связи с использованием jQuery Mobile.

Событие `pageinit`

Библиотека jQuery Mobile автоматически улучшает страницы, регистрируя свои функции для обработки события `ready` jQuery, на которое мы опирались в предыдущих частях. Если вы намереваетесь включить в документ пользовательский JavaScript-код, проследите за тем, чтобы этот код выполнялся лишь после обработки документа средствами jQuery Mobile. Это означает, что необходимо ожидать наступления другого события — `pageint`. Это событие определено в jQuery Mobile и происходит после того, как jQuery Mobile завершает инициализацию документа. Какого-либо удобного метода, аналогичного тому, который предусмотрен для события `ready`, не существует, поэтому для связывания своей функции с событием нужно воспользоваться методом `bind()`, как показано в листинге 26.4.

Листинг 26.4. Использование события `pageinit`

```
<!DOCTYPE html>

<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
```

```
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $('button').click(function() {
                console.log("Нажатие кнопки")
            })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="header">
                <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><button>Нажми меня</button></p>
        </div>
   </div>
</body>
</html>
```

Аргументами метода `bind()` являются имя интересующего вас события и функция, которая должна быть вызвана в ответ на наступление события. Эта функция будет выполняться лишь в том случае, если событие относится к элементу или элементам, которые были выбраны и к которым был применен метод `bind()`.

Здесь метод `bind()` используется для регистрации функции, которая будет выполняться при наступлении события `pageinit`. В тело функции помещены инструкции, которые выполнятся после загрузки и обработки документа. В данном случае мы выбираем в документе элемент `button` с помощью jQuery и используем метод `click()` для регистрации другой функции, вызов которой инициируется щелчком на кнопке, как это делалось на протяжении всей книги.

Совет. Обратите внимание, что новый элемент `script` *предшествует* элементу `script`, импортирующему JavaScript-библиотеку jQuery Mobile в документ. Это несущественно для события `pageinit`, но требуется для события `mobileinit`, которое используется для изменения некоторых установочных параметров jQuery Mobile (более подробно об этом речь идет в главе 27). Я пришел к выводу, что даже в тех случаях, когда обрабатывается одно лишь событие `pageinit`, целесообразно вставлять пользовательский JavaScript-код в документ перед элементами, импортирующими библиотеку jQuery Mobile.

События касаний

Для событий `touch`, происходящих в браузере, существует спецификация, однако она охватывает лишь низкоуровневые события, поскольку число возможных моделей интерактивного взаимодействия посредством касаний очень велико. Например, некоторые устройства поддерживают одновременное определение координат нескольких точек касания (так называемая функция *мультитач*, обеспечивающая распознавание множественных касаний, и соответственно — *мультитач-*

устройства), и в этом случае жесты, выполняемые одновременно несколькими пальцами (мультитач-жесты), могут интерпретироваться самыми разными способами. Низкоуровневые события касаний описаны в табл. 26.3.

Таблица 26.3. Стандартные события касаний

Событие	*Описание*
`touchstart`	Происходит, когда пользователь касается экрана пальцем. Для устройств, поддерживающих множественные касания (мультитач-устройств), это событие происходит независимо для каждой точки касания
`touchend`	Происходит, когда пользователь убирает палец с экрана
`touchmove`	Происходит, когда пользователь перемещает палец, не отрывая его от экрана
`touchcancel`	Происходит в момент прерывания последовательности прикосновений. Смысл этого события зависит от устройства, но обычно оно означает, что палец пользователя выходит за пределы экрана

Ответственность за интерпретацию этих событий и придание им определенного смысла возлагается на веб-разработчика. Это весьма непростая задача, чреватая многими ошибками, и я рекомендую вам по возможности не пытаться решать ее самостоятельно и довериться в этом библиотеке jQuery Mobile, о чем мы вскоре поговорим.

Совет. Если вы хотите узнать больше о событиях касания, обратитесь к спецификации по адресу `http://www.w3.org/TR/touch-events`. В ней содержится полное описание событий и свойств, предоставляемых для получения подробных сведений о каждом виде интерактивного взаимодействия посредством касаний.

Большинство веб-сайтов проектировалось без учета событий касания. Чтобы обеспечить поддержку максимально широкого круга сценариев, используемых на веб-сайтах, мобильные браузеры синтезируют события мыши на основе событий касания. Это означает, что браузер запускает события касания, а затем генерирует соответствующие (фиктивные) события мыши, которые представляют те же действия, но так, словно они были выполнены традиционным способом с помощью мыши. Пример полезного сценария, в котором продемонстрировано, как это можно сделать, приведен в листинге 26.5.

Листинг 26.5. Отслеживание событий касания и синтезированных событий мыши

```
<!DOCTYPE html>
<html>
<head>
    <title>Тестирование событий</title>
    <meta name="viewport"
        content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <style type="text/css">
        table {border-collapse: collapse;
            border: medium solid black; padding: 4px}
        #placeholder {text-align: center}
        #countContainer * {display: inline; width:50px}
```

```
        th {width: 100px}
    </style>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            var eventList = [
                "mousedown", "mouseup", "click", "mousecancel",
                "touchstart", "touchend", "touchmove",
                "touchcancel"]

            for (var i = 0; i < eventList.length; i++) {
                $('#pressme').bind(eventList[i], handleEvent)
            }

            $('#reset').bind("tap", function() {
                $('tbody').children().remove();
                $('#placeholder').show();
                startTime = 0;
            })
        });

        startTime = 0;
        function handleEvent(ev) {
            var timeDiff = startTime == 0 ? 0 :
               (ev.timeStamp - startTime);
            if (startTime == 0) {
                startTime = ev.timeStamp
            }
            $('#placeholder').hide();
            $('<tr><td>' + ev.type + '</td><td>' + timeDiff +
                '</td></tr>').appendTo("tbody");
        }
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="content">
            <div id="tcontainer" class="ui-grid-a">
                <div class="ui-block-a">
                    <button id="pressme">Нажми меня</button>
                    <button id="reset">Сброс</button>
                </div>
                <div class="ui-block-b">
                    <table border=1>
                        <thead><tr><th>Событие</th><th>Время</th>
                            </tr>
                            <tr id="placeholder">
                                <td colspan=2>Нет событий</td><tr>
                        </thead>
                        <tbody></tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере мы имеем дело с двумя кнопками и одной таблицей. Привязка событий к кнопке Нажми меня осуществлена таким образом, чтобы после щелчка на кнопке в таблице отображалась информация о событиях касаний и мыши. Для каждого события выводится его тип и количество миллисекунд, истекших с момента наступления последнего события. Щелчок на кнопке Сброс приводит к очистке полей таблицы и обнулению таймера. Результат представлен на рис. 26.4.

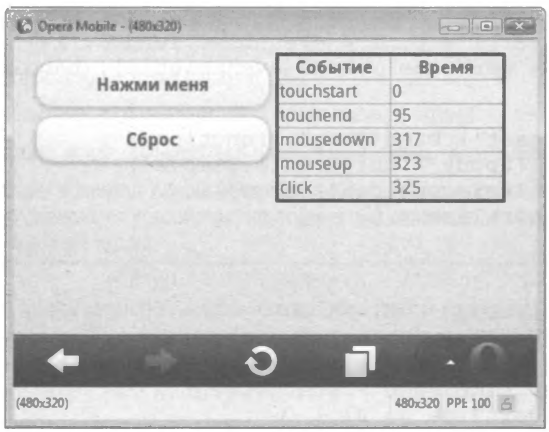


Рис. 26.4. Последовательность событий касаний и мыши

Данные о последовательности событий и их развертывании во времени после щелчка на кнопке в браузере Opera Mobile представлены в табл. 26.4.

Таблица 26.4. Последовательность событий в браузере Opera Mobile

Событие	*Относительное время*
`touchstart`	0
`touchend`	95
`mousedown`	317
`mouseup`	323
`click`	325

Данные таблицы свидетельствуют о том, что сначала происходят события `touchstart` и `touchend`, соответствующие моментам касания и отрыва пальца от экрана. Уже после этого браузер генерирует события `mousedown`, `mousedown` и `click`. Обратите внимание на то, что между моментами наступления событий `touchend` и `mousedown` наблюдается заметная задержка, длительность которой составляет около 300 мс. Такой задержки достаточно для того, чтобы поставить под сомнение целесообразность использования искусственных событий, поскольку взаимодействие пользователя с экраном посредством касаний будет тормозить работу вашего приложения. Эта проблема свойственна не всем браузерам, но достаточно распространена, чтобы о ней стоило упомянуть, и поэтому я рекомендую всегда тестировать длительность задержек в браузерах, которые были выбраны вами в качестве целевых.

Использование методов jQuery Mobile для работы с жестами

Библиотека jQuery Mobile упрощает работу с событиями двояким образом. Во-первых, в ней предусмотрен набор событий жестов, которые происходят в ответ на определенную последовательность низкоуровневых событий касания, а это означает, что вы не должны самостоятельно анализировать такие последовательности для интерпретации смысла жестов, совершаемых пользователями. Эти события описаны в табл. 26.5.

Таблица 26.5. Стандартные события жестов

Событие	*Описание*
`tap`	Происходит, когда пользователь касается пальцем экрана, а затем быстро убирает его
`taphold`	Происходит, когда пользователь касается экрана, удерживает палец на месте в течение примерно одной секунды, а затем убирает его
`swipe`	Происходит, когда пользователь перемещает палец по экрану на расстояние по крайней мере 30 пикселей при изменении положения точки касания по вертикали менее чем на 20 пикселей, совершая этот жест примерно за одну секунду
`swipeleft`	Происходит, когда пользователь перемещает палец по экрану справа налево
`swiperight`	Происходит, когда пользователь перемещает палец по экрану слева направо

Эти события значительно упрощают обработку базовых жестов. Добавление перечисленных событий в предыдущий пример представлено в листинге 26.6.

Листинг 26.6. Добавление событий жестов jQuery Mobile в пример с хронометрированием событий

```
...
<script type="text/javascript">
    $(document).bind("pageinit", function() {
        var eventList = [
            "mousedown", "mouseup", "click", "mousecancel",
            "touchstart", "touchend", "touchmove", "touchcancel",
            "tap", "taphold", "swipe", "swipeleft", "swiperight"]

        for (var i = 0; i < eventList.length; i++) {
            $('#pressme').bind(eventList[i], handleEvent)
        }

        $('#reset').bind("tap", function() {
            $('tbody').children().remove();
            $('#placeholder').show();
            startTime = 0;
        })
    });

    startTime = 0;
    function handleEvent(ev) {
        var timeDiff = startTime == 0 ? 0 :
            (ev.timeStamp - startTime);
        if (startTime == 0) {
            startTime = ev.timeStamp
        }
        $('#placeholder').hide();
```

```
        $('<tr><td>' + ev.type + '</td><td>' + timeDiff +
            '</td></tr>').appendTo("tbody");
    }
</script>

...
```

На рис. 26.5 показано, что происходит в результате щелчка на кнопке.

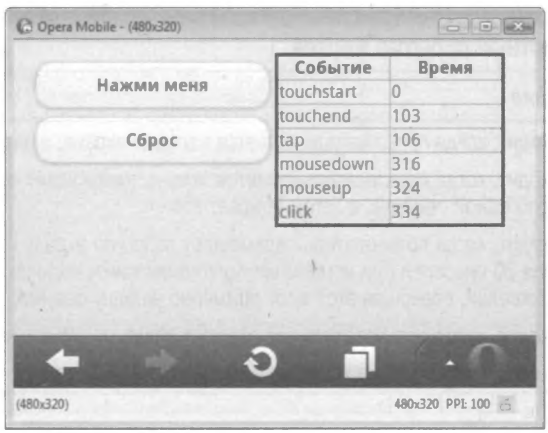


Рис. 26.5. Добавление событий жестов jQuery Mobile в пример с хронометрированием событий

Данные о последовательности событий представлены в более удобной для восприятия форме в табл. 26.5. Поскольку эти события были порождены щелчком на кнопке, наблюдается только одно событие жестов — `tap`. Обращаю ваше внимание на важную деталь: событие `tap` наступает очень быстро, через несколько миллисекунд после отпускания кнопки мыши.

Таблица 26.6. Последовательность событий в браузере Opera Mobile

Событие	*Относительное время*
`touchstart`	0
`touchend`	103
`tap`	**106**
`mousedown`	316
`mouseup`	324
`click`	334

Удобным свойством событий жестов является то, что jQuery Mobile генерирует их даже в браузерах, которые не поддерживают событий касаний, или на устройствах, не имеющих сенсорных интерфейсов. Результат выполнения примера в настольном браузере Google Chrome представлен на рис. 26.6.

В более удобном для чтения виде эти данные представлены в табл. 26.7.

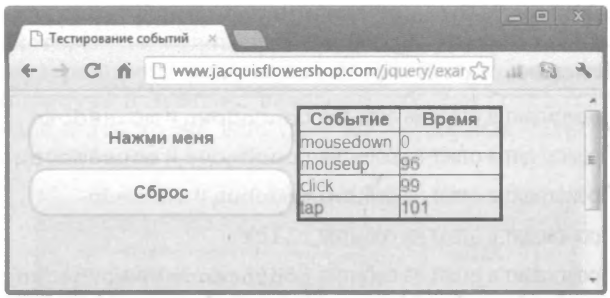


Рис. 26.6. Последовательность событий в настольном браузере

Таблица 26.7. Последовательность событий в браузере Google Chrome

Событие	*Относительное время*
`mousedown`	0
`mouseup`	96
`click`	99
`tap`	**101**

В этой последовательности не только отсутствуют события `touchstart` и `touchend`, что не является сюрпризом, но и порядок событий другой (поскольку в данном случае события мыши реальные, а не имитируемые). Но, как бы то ни было, событие `tap` запускается, причем спустя всего лишь две миллисекунды после наступления события `click`.

В мобильных приложениях я использую событие `tap` вместо события `click`, поскольку оно позволяет избежать проблем с задержками и генерируется даже на платформах, не являющихся сенсорно-ориентированными.

Использование виртуальных событий мыши jQuery Mobile

В обычных браузерах события мыши не имитируются, и веб-приложение, которое работает как на сенсорных устройствах, так и на устройствах, не поддерживающих эту функциональность, должно прослушивать и события мыши, и события касания. В случае мобильных браузеров, которые должны синтезировать эти события, для каждого взаимодействия предоставляются как события касания, так и события мыши. Для упрощения обработки этих событий в jQuery Mobile определен набор виртуальных событий мыши. Если вы зарегистрируете функции для обработки этих событий, то jQuery Mobile проследит за удалением дубликатов и обеспечит гарантированный запуск соответствующих событий, независимо от наличия или отсутствия поддержки касаний. Виртуальные события мыши описаны в табл. 26.8.

Таблица 26.8. Виртуальные события мыши

Событие	*Описание*
`vmouseover`	Происходит в ответ на событие `mouseover` (для события `touch` аналогичный эквивалент отсутствует, поскольку палец пользователя не находится в постоянном контакте с экраном)

Окончание табл. 26.8

Событие	Описание
vmousedown	Происходит в ответ на события touchdown и mousedown
vmousemove	Происходит в ответ на события touchmove и mousemove
vmouseup	Происходит в ответ на события touchup и mouseup
vclick	Происходит в ответ на событие click
vmousecancel	Происходит в ответ на события touchcancel и mousecancel

Эти события генерируются таким образом, что их последовательность аналогична последовательности событий мыши даже на сенсорных устройствах. Чтобы пояснить, что именно имеется в виду, в пример с хронометрированием событий добавлены некоторые виртуальные события, как показано на рис. 26.7.

Листинг 26.7. Добавление виртуальных событий jQuery Mobile в пример с хронометрированием событий

```
...
<script type="text/javascript">
    $(document).bind("pageinit", function() {
        var eventList = [
            "mousedown", "mouseup", "click", "mousecancel",
            "touchstart", "touchend", "touchmove", "touchcancel",
            "tap", "taphold", "swipe", "swipeleft", "swiperight",
            "vmouseover", "vmousedown", "vmouseup", "vclick",
            "vmousecancel"]

        for (var i = 0; i < eventList.length; i++) {
            $('#pressme').bind(eventList[i], handleEvent)
        }

        $('#reset').bind("tap", function() {
            $('tbody').children().remove();
            $('#placeholder').show();
            startTime = 0;
        })
    });

    startTime = 0;
    function handleEvent(ev) {
        var timeDiff = startTime == 0 ? 0 :
           (ev.timeStamp - startTime);
        if (startTime == 0) {
            startTime = ev.timeStamp
        }
        $('#placeholder').hide();
        $('<tr><td>' + ev.type + '</td><td>' + timeDiff +
            '</td></tr>').appendTo("tbody");
    }
</script>
...
```

Когда пользователь касается экрана, jQuery Mobile генерирует события vmouseover и vmousedown. В исключительно сенсорной среде эти события ничего не означают.

Если вы пишете кроссплатформенное приложение, то, вероятно, предусматриваете выполнение некоторых действий при наведении указателя мыши на определенный элемент в браузере настольного компьютера. Запуск искусственно генерируемого события `vmouseover` в ответ на реальное событие `touchstart` позволяет беспрепятственно выполнить то же действие на сенсорных устройствах. Результат представлен на рис. 26.7.

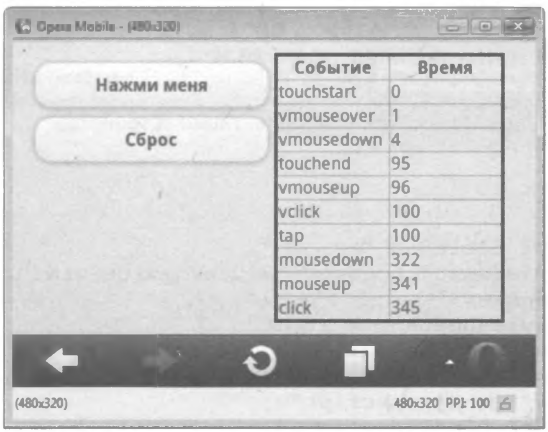


Рис. 26.7. Добавление виртуальных событий в пример с хронометрированием событий

Таблица 26.9. Последовательность событий в браузере Opera Mobile

Событие	*Относительное время*
`touchstart`	0
`vmouseover`	**1**
`vmousedown`	**4**
`touchend`	95
`vmouseup`	96
`vclick`	**100**
`tap`	100
`mousedown`	322
`mouseup`	341
`click`	345

В более удобном для чтения виде эти данные представлены в табл. 26.9.

Предупреждение. Важно не делать никаких предположений относительно взаимной очередности реальных и виртуальных событий. Дело в том, что последовательность событий на устройствах, не являющихся сенсорными, будет другой. Взаимный порядок следования виртуальных событий остается для этих устройств таким же, но конфигурация последовательности перемежающихся с ними реальных событий может измениться.

Реагирование на изменение ориентации устройства

Большинство мобильных браузеров поддерживает событие `orientationchange`, которое происходит всякий раз, когда ориентация устройства меняется на 90°. Чтобы облегчить вам жизнь, jQuery Mobile синтезирует событие `orientationchange` в тех случаях, когда оно не поддерживается браузером. Это достигается за счет отслеживания размеров окна и проверки величины отношения новых значений его высоты и ширины. Пример, в котором продемонстрировано, как можно реагировать на это событие, приведен в листинге 26.8.

Листинг 26.8. Реагирование на изменение ориентации устройства

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $(window).bind("orientationchange", function(e) {
                $('#status').text(e.orientation)
            })
            $('#status').text(jQuery.event.special
                .orientationchange.orientation())
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="header">
                <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <p>Ориентация устройства: <b><span id=status></span>
                </b></p>
        </div>
   </div>
</body>
</html>
```

Для привязки к событию `orientationchange` вы должны выбрать объект `window`. В этом примере индикатором изменения ориентации служит изменение текста элемента `span`. Соответствующая информация доступна через свойство `orientation` объекта `Event`, передаваемого обработчику события.

Кроме того, jQuery Mobile предоставляет метод для определения текущей ориентации: `jQuery.event.special.orientationchange.orientation()`

В примере этот метод используется для установки содержимого элемента `span`, поскольку событие `orientationchange` запускается не при обработке страницы, а при последующей переориентации устройства.

Если реального мобильного устройства для тестирования данного примера у вас нет под рукой, можете воспользоваться одним из эмуляторов, описанных далее. Большинство из них способны имитировать вращение, вызываемое нажатием определенной комбинации клавиш или нажатием кнопки. В эмуляторе Opera Mobile, которым я пользуюсь, это достигается одновременным нажатием клавиш <Ctrl+Alt+R>, что дает результат, представленный на рис. 26.8.

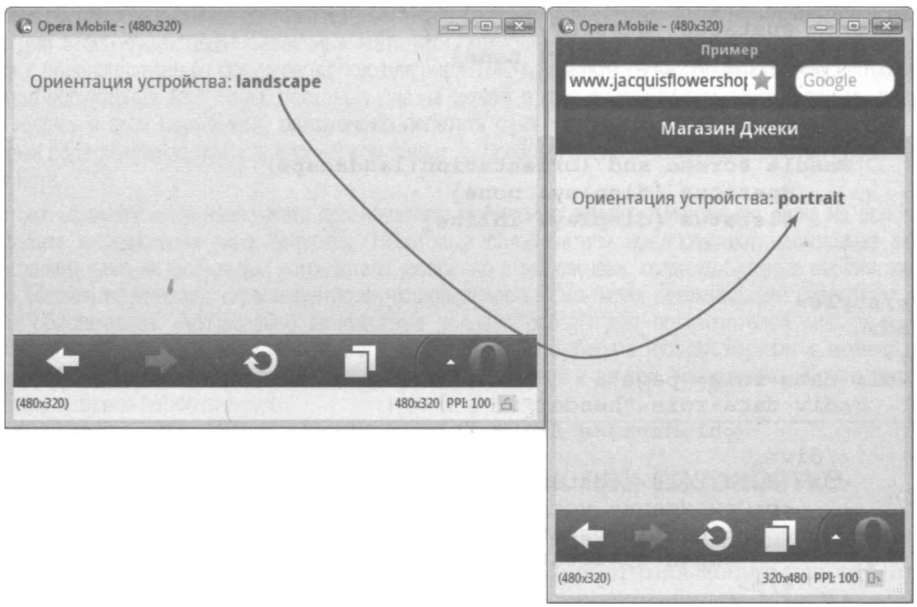


Рис. 26.8. Добавление виртуальных событий в пример с хронометрированием событий

Синтезированное jQuery Mobile событие позволяет получить тот же результат в браузере, который не поддерживает изменение пространственной ориентации. Это достигается путем изменения размеров его окна. В данном случае ориентация окна определяется значениями его ширины и высоты.

Использование медиазапросов для управления ориентацией

Используя событие `orientationchange`, можно реагировать на изменение ориентации с помощью JavaScript. Альтернативный подход состоит в применении отличающихся стилей CSS к элементам с разной ориентацией посредством модуля *медиазапросов CSS3* (CSS3 Media Queries). Пример того, как это можно сделать, приведен в листинге 26.9.

Листинг 26.9. Реагирование на изменение ориентации с помощью технологии CSS3 Media Queries

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
```

```
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">

        @media screen and (orientation:portrait) {
            #pstatus {display: inline}
            #lstatus {display: none}

        }

        @media screen and (orientation:landscape) {
            #pstatus {display: none}
            #lstatus {display: inline}
        }

    </style>
</head>
<body>
    <div data-role="page">
        <div data-role="header">
                <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <p>Ориентация устройства:
                <span id=pstatus><b>Портретная</b></span>
                <span id=lstatus><b>Альбомная</b></span>
            </p>
        </div>
   </div>
</body>
</html>
```

Медиазапросы CSS3 позволяют определять наборы так называемых условных стилей, т.е. стилей, которые применяются в зависимости от определенных обстоятельств, в данном случае в зависимости от того, какую ориентацию имеет страница: портретную или альбомную. Используя CSS-свойство `display` для отображения или сокрытия элементов, можно получить тот же результат, который был достигнут в предыдущем примере с помощью JavaScript. При этом никакой необходимости в генерации искусственных событий не возникает. Запросы ориентации с помощью средства Media Queries работают одинаково хорошо как для настольных, так и для мобильных браузеров.

Работа с мобильными устройствами

Между процессами разработки приложений для мобильных устройств и настольных компьютеров имеются существенные отличия. Информация по этому вопросу вместе с соответствующими рекомендациями приведена в следующих разделах, в которых также описаны основные проблемы, с которыми приходится сталкиваться разработчикам мобильных приложений.

Распознавание мобильных устройств

Если предлагаемое вами приложение предназначено как для настольных компьютеров, так и для мобильных устройств, у вас может возникнуть желание улучшить имеющийся интерфейс. Обычный подход к обеспечению нормальной работы приложения с разными браузерами ориентирован на то, чтобы браузеры настольных компьютеров получали приложение с интерфейсом jQuery UI, а мобильные браузеры — с интерфейсом jQuery Mobile.

Главная трудность связана с распознаванием типов браузеров, выполняющихся на мобильных устройствах[1]. Для этого существуют различные методики, которые реализуются на стороне сервера и сводятся к перенаправлению браузера на подходящие HTML-документы. Я не собираюсь углубляться в детальное обсуждение этой темы, поскольку она не входит в круг основных тем книги. Если вы впервые столкнулись с этой проблемой, рекомендую посетить сайт `http://wurfl.sourceforge.net`, который содержит полезный серверный компонент, способный распознавать большинство современных устройств.

Не стоит автоматически навязывать пользователю мобильную версию приложения лишь на основании результата определения типа браузера. Некоторые пользователи предпочитают настольные версии приложений даже на мобильных устройствах, особенно в тех случаях, когда мобильные версии, как это часто бывает, предлагают ограниченную функциональность. Суть моих рекомендаций сводится к тому, чтобы обеспечивать максимально комфортные условия работы для пользователей мобильных устройств, предоставляя им возможность несложным образом выбирать нужный вариант в момент посещения сайта и упрощая переход от одной версии приложения к другой, независимо от того, какая из них была выбрана первоначально.

Как избежать двух основных ошибок при разработке мобильных приложений

Самые опасные ловушки, подстерегающие разработчика мобильных веб-приложений, — это *необоснованные предположения* и *нереалистическое моделирование*. Прежде чем двигаться дальше, я проанализирую обе эти ловушки и приведу примеры некоторых типичных ситуаций, рассмотрение которых поможет вам избежать некоторых распространенных ошибок.

Избегайте необоснованных предположений

Рынок мобильных устройств чрезвычайно динамичен, сравнительно незрел и очень плохо определен. Обычно при создании веб-приложений для настольных компьютеров исходят из каких-то начальных предположений (о которых часто ничего не говорят). За многие годы постепенно сложился некий всеобщий уровень ожиданий относительно минимального разрешения экрана, поддержки JavaScript, доступности определенных плагинов, не говоря уже о безоговорочно предполагаемой возможности указания объектов с помощью мыши и ввода текста с помощью клавиатуры.

[1] Сейчас все больше и больше говорят об альтернативных подходах — распознавании объектов (object detection) и распознавании возможностей (feature detection). Информация о типе браузера (так называемая UA-строка), предоставляемая серверу, может быть легко изменена пользователем и поэтому не может считаться полностью достоверной. Указанные альтернативные подходы обеспечивают получение более надежных сведений о том, на что действительно способен браузер. В конце концов, именно для этого и осуществляется распознавание типа браузера. — *Примеч. ред.*

Вместе с тем такие предположения не всегда оправданны. Если, например, вы исходите из того, что приложению будет всегда доступен JavaScript, то тем самым из круга ваших потенциальных клиентов выпадают пользователи, которые не хотят (или не могут) разрешать выполнение сценариев JavaScript в своем браузере. Возможно, вы сочтете допустимым для себя компромиссом ситуацию, когда большинство пользователей при желании могут активизировать JavaScript на своих компьютерах, а от пользователей, браузеры которых не удовлетворяют требованиям вашего приложения, вы просто отказываетесь.

На рынке мобильных устройств в силу его фрагментарности наблюдается еще худшая ситуация. Мир настольных компьютеров предстает перед нами разнообразным, но у всех компьютеров Mac, Windows и Linux есть много общего. О мобильных устройствах подобного не скажешь, и следствием любых предположений об экранном разрешении, сетевых подключениях и методах ввода информации в пользовательском браузере будет автоматическое исключение целых сегментов рынка.

На iPhone свет клином не сошелся

Одним из наиболее пагубных допущений, с которыми мне приходилось сталкиваться (причем довольно часто), является предположение о том, что целевым рынком следует считать рынок iPhone. Действительно, iPhone пользуется небывалым успехом, но рынок мобильных устройств не ограничивается только им, да и разные модели iPhone отличаются друг от друга. Типичным целевым значением экранного разрешения является 320×480 пикселей, которое было характерным еще для ранних моделей iPhone. С таким разрешением работают многие устройства, но при этом постоянно растет число устройств с другим разрешением экрана. Использование какого-то фиксированного значения в приложении будет просто отсеивать тех пользователей, устройства которых имеют небольшой экран, и вызывать недовольство пользователей, которые не пожалели денег на устройства с более высоким экранным разрешением.

В мире существуют не только телефоны

Другое распространенное допущение состоит в том, что целевым рынком считают мобильные телефоны и тем самым игнорируют успехи рынка планшетных устройств. Планшеты не только обладают более высоким разрешением экрана, но и отличаются от других устройств тем, как с ними обычно обращаются и как их используют. Чтобы понять, что именно я имею в виду, загляните в любое кафе и понаблюдайте за посетителями. В соответствии с моими (абсолютно ненаучными, но регулярно подтверждающимися) наблюдениями, чем больше размер планшета, тем неудобнее держать его в руках и тем чаще люди вынуждены искать для него какую-нибудь опору. Это означает, что планшет не совсем устойчив, в результате чего при проведении пальцем по экрану планшет слегка покачивается (нарушается точность), а большая часть экрана оказывается закрытой (поскольку рука и ладонь находятся над планшетом).

Моя точка зрения такова, что сама природа мобильных устройств во многом диктует, как ими пользоваться и какие типы интерактивного взаимодействия уместны и желательны. Наилучшим способом определить это является наблюдение за людьми, взаимодействующими с самыми различными устройствами. Если у вас есть время и деньги, то фантастические возможности в этом смысле обеспечила бы лаборатория по тестированию удобства использования. Но даже если вы испытываете дефицит как времени, так и денег, то обед в кафе может обогатить вас множеством идей.

Не весь мир сенсорно-ориентированный

Не все мобильные устройства имеют сенсорные экраны. Одни из них снабжены крохотными мышью и клавиатурой, в других используется несколько методов ввода. Одна из моих тестовых машин — небольшой ноутбук, который легко превращается в планшет. Он оборудован сенсорным экраном, а также полноценными клавиатурой и мышью. Пользователи рассчитывают на то, что будут иметь возможность использовать наиболее приемлемый из методов ввода, доступных для них, и поэтому допущения о том, что какие-то методы ввода им не нужны, приведут лишь к их неудовлетворенности (именно поэтому я редко пользуюсь своим комбинированным ноутбуком-планшетом).

Полоса пропускания не бесплатна и не беспредельна

Цена сетевых подключений колеблется в зависимости от того, какие виды деятельности выполняются пользователем в сети. В настоящее время сетевые операторы стремятся вкладывать средства для обеспечения пропускной способности, достаточной для удовлетворения запросов потребителей, особенно в густонаселенных городских районах. Когда-нибудь стоимость пропускной способности упадет до незначительного уровня, а доступная полоса пропускания увеличится, но в настоящее время сетевые операторы взимают дополнительную плату за доступ к данным и устанавливают предельные объемы данных, которые пользователям разрешается загружать в течение месяца.

Исходить из того, что пользователь готов согласиться на обработку существенных объемов данных вашим веб-приложением — опасная практика. В большинстве случаев пользователи не испытывают к вашему приложению особой привязанности, как вам того хотелось бы. Возможно, это ранит ваше самолюбие, но в большинстве случаев это именно так. Для вас весь мир заключен в вашем приложении, и это вполне понятно, но для пользователей оно является всего лишь одним из многих доступных.

В главе 27 будет показано, что jQuery Mobile может предварительно загружать содержимое для веб-приложения еще до того, как оно понадобится пользователю. Эта возможность просто замечательна, но ею следует пользоваться осторожно, поскольку подразумевается, что пользователь согласен тратить часть полосы пропускания на передачу содержимого, необходимость в котором может никогда не возникнуть. То же самое можно сказать о частом автоматическом обновлении данных. Пользуйтесь этими ресурсами экономно, тщательно взвешивая все факторы, и только тогда, когда пользователи явным образом указывают, что вашему приложению разрешено интенсивно использовать их квоту сетевого трафика.

Точно так же никогда не делайте допущений относительно доступной для мобильного устройства скорости передачи данных. Позаботьтесь о рациональном использовании таких "пожирателей" ресурсов, как изображения и видео. Некоторые пользователи будут иметь каналы связи, обеспечивающие возможность быстрой загрузки подобного содержимого, а некоторые — не будут, поэтому я всегда предусматриваю возможность выбора конкретного варианта загрузки содержимого, требующего незначительной ширины полосы пропускания.

Вы всегда должны быть готовы к тому, что в какой-то момент сеть окажется недоступной. Мне часто приходилось ездить на поезде, и каждый раз, когда поезд въезжал в тоннель, сеть пропадала. Хорошо написанное веб-приложение должно справляться с проблемами подключения, сообщать о них пользователям и корректно возобновлять работу, когда сеть вновь становится доступной. Как это ни прискорбно, большинство веб-приложений не относятся к этой категории.

Избегайте нереалистического моделирования и тестирования

Большое разнообразие мобильных устройств означает необходимость тщательного тестирования. Работа с реальными мобильными устройствами, начиная с ранних стадий обработки, кому-то может не нравиться. Сетевые запросы маршрутизируются через сотовые телефонные сети, а для этого требуется, чтобы компьютеры разработчиков были общедоступными. Для некоторых мобильных устройств предусмотрены режимы разработчика, но здесь есть свои недостатки.

Если говорить кратко, то для того, чтобы приступить к разработке, нужна имитация рабочей среды, обеспечивающей возможность быстрого создания и тестирования приложения удобными способами в условиях, когда эта среда не связана с внешним миром. К счастью, существуют эмуляторы, предоставляющие все необходимые для этого средства. Некоторые из доступных для выбора эмуляторов описаны далее, но в целом они делятся на две категории.

К первой категории относятся эмуляторы, в которых реальный мобильный браузер переносится на другую платформу. При этом добиваются того, чтобы все факторы, связанные с браузером, были по возможности максимально приближены к реальным условиям. Ко второй категории относятся эмуляторы, в основе которых лежит тот факт, что в большинстве браузеров один и тот же движок визуализации используется как для мобильных, так и для настольных устройств. К примеру, если требуется оценить в общих чертах, как браузер iPhone обработает документ, можно использовать браузер Apple Safari, поскольку у них имеется много общих корней. Эмулятор — это нечто большее, чем просто визуальная оболочка и ограничение разрешения экрана, примененные к движку визуализации настольной системы.

Полезными могут быть оба подхода, и оба они заслуживают внимания. Я часто использую их на ранних стадиях разработки мобильных продуктов. Но сразу же после того, как базовая функциональность реализована, я начинаю тестировать приложение на реальных устройствах, а когда приближаюсь к завершению работы над проектом, то сразу же перехожу к использованию реальных устройств и вообще не использую эмуляторы.

Объясняется это тем, что эмуляторы имеют два серьезных недостатка. Первый из них — эмуляция не бывает на 100% точной. Даже наилучшим эмуляторам не всегда удается представить содержимое так, как оно будет представлено в реальном устройстве с использованием того же браузера. Второй, и, по моему мнению, более серьезный недостаток, заключается в их неспособности имитировать средства сенсорного ввода.

Работа сенсорно-ориентированных браузеров на настольных ПК обеспечивается с помощью мыши, но мышь не может в полной мере воссоздать эффекты, которые достигаются касанием экрана пальцем. При эмуляции на настольном компьютере исчезают три фактора, присущие работе с сенсорными устройствами: *тактильные ощущения*, *заслонение экрана* и *недостаточная точность*.

Отсутствие тактильных ощущений

Отсутствие тактильных ощущений означает, что вы не сможете достоверно оценить, как будет *осязаться* веб-приложение в процессе его использования. Прикосновение к гладкой поверхности экрана или проведение по ней пальцем — довольно необычные действия. Если приложение реагирует на эти действия надлежащим образом, то работа с ним доставляет удовольствие. Любые задержки реакции на ввод или неправильная реконструкция интерактивного взаимодействия

пользователя с приложением посредством касаний может вызвать только раздражение. Мышь не в состоянии обеспечить обратную связь, равноценную той, которая создается путем обработки касаний.

Отсутствие заслонения экрана

Ранее уже затрагивалась проблема заслонения экрана. Когда вы используете сенсорные устройства, ваши палец и рука закрывают часть экрана, что особенно сказывается в случае небольших устройств. Проектируя веб-приложение для подобных устройств, обязательно нужно учитывать этот фактор. Необходимо внимательно следить за размещением элементов управления, чтобы при касании некоторого участка пользователь мог видеть, что происходит на остальной части экрана. Кроме того, следует иметь в виду, что примерно 10% населения — левши, и поэтому для них заслоняться будут другие части экрана. Лишь управление кнопками и ссылками путем касания их собственными пальцами дает истинное понимание того, просто ли пользоваться приложением.

Совет. Если вы послушаетесь моего совета и зайдете в кафе, чтобы понаблюдать за пользователями мобильных устройств, ищите пользователей, следующих описанному ниже шаблону поведения. Эти пользователи касаются экрана, затем полностью убирают от него руку примерно на секунду и только после этого вновь опускают руку для выполнения очередного жеста. Очень часто это служит признаком того, что в приложении виджеты размещены так, что результат визуальной обратной связи, обусловленной выполнением некоторого действия, отображается в области экрана, заслоненной рукой пользователя. Пользователь должен убрать руку от экрана, чтобы оценить полученный результат, прежде чем выполнить очередной жест (довольно утомительная и неприятная процедура).

Недостаточная точность

Вопрос точности относится к разряду тех, о которых постоянно приходится помнить. С помощью мыши пользователь может указать любой объект на экране с исключительной точностью. Используя современную модель мыши и немного попрактиковавшись, можно добиться точности в один пиксель. Человеческий палец такую точность не обеспечит, и самое большее, на что можно рассчитывать, — это "приблизительно в области объекта". Отсюда следует, что нужно создавать виджеты, которые легко выбирать, и макеты страниц, которые учитывают эту погрешность. В случае эмулятора ощутить, насколько точно удается указать виджет, невозможно. В этом отношении мышь не является хорошим помощником. Чтобы понять, с чем в действительности столкнутся пользователи, нужно будет выполнить тесты на реальных устройствах в широком диапазоне различных разрешений экрана и его размеров. Эта информация позволит получить ценные подсказки относительно размера и плотности расположения виджетов на создаваемых веб-страницах.

Использование эмуляторов мобильных браузеров

Даже при наличии описанных ограничений эмуляторы мобильных устройств могут быть незаменимыми при разработке приложений. В этом разделе описаны некоторые доступные возможности и проанализированы доводы "за" и "против" каждого из них.

Использование эмулятора Android

В комплект разработчика (SDK) для платформы Android входит эмулятор. Он довольно точно воспроизводит результаты работы браузера на реальных устройствах, но с точки зрения скорости оставляет желать лучшего. Дело в том, что объек-

том эмуляции является не только браузер, но и вся операционная система. Поэтому эмулятор работает крайне медленно и не всегда реагирует на запросы, причем даже на мощных настольных компьютерах, что не может не раздражать. На рис. 26.9 представлен образец документа в браузере, который входит в SDK, доступный для загрузки по адресу `http://developer.android.com/sdk`.

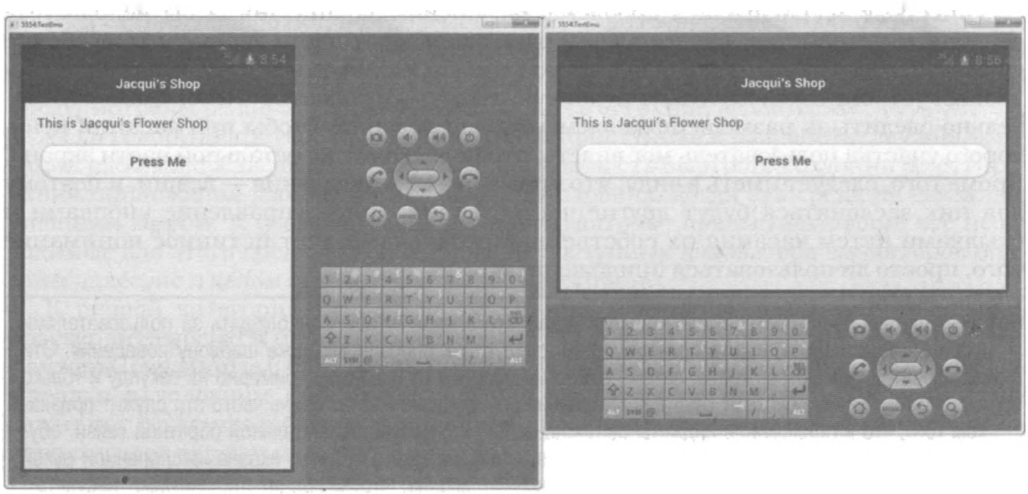


Рис. 26.9. Использование эмулятора из Android SDK

Использование эмулятора iPhone

Компания Apple включает эмулятор для iOS как часть своего пакета разработки Xcode, бесплатно распространяемый через интернет-магазин Mac App Store. Разумеется, для выполнения Xcode нужно иметь компьютер Mac, и поэтому я (как пользователь Windows и Linux) никогда не работал с этим пакетом.

Использование эмулятора Opera Mobile

Это один из тех эмуляторов, которые я чаще всего использую, поскольку он позволяет имитировать устройства с экранами различных размеров, включая планшеты, а также альбомную ориентацию. Браузер Opera Mobile широко используется и хорошо (хотя и не безукоризненно) справляется с работой по размещению содержимого на странице. Некоторые возможности jQuery Mobile, такие как навигационные переходы (которые описаны в главе 27), не поддерживаются. Простой пример документа в эмуляторе Opera Mobile представлен на рис. 26.10.

Удобно то, что для отладки в эмуляторе можно воспользоваться отладчиком, встроенным в настольную версию браузера Opera. Настройка отладчика в этом случае займет какое-то время, но оно себя окупит. Эмулятор Opera Mobile можно загрузить бесплатно по следующему адресу:

`http://www.opera.com/developer/tools/mobile`

Использование эмулятора Firefox Mobile

Эмулятор Firefox Mobile доступен для загрузки по адресу `www.mozill;a.org/mobile`. Он неплохо имитирует работу браузера, выполняющегося на реальных мобильных устройствах, однако менее эффективен в отношении установки различных значений

экранного разрешения по сравнению с эмулятором браузера Opera и не поддерживает альбомные ориентации. Этот эмулятор является бесплатным и поддерживается широким рядом настольных платформ. Пример документа в окне эмулятора Firefox Mobile представлен на рис. 26.11.

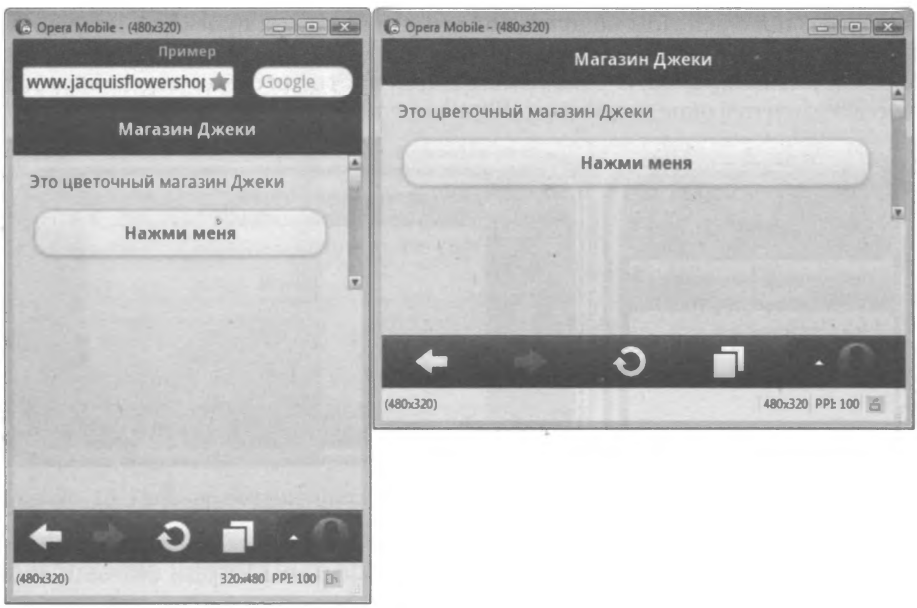


Рис. 26.10. Пример документа в эмуляторе Opera Mobile

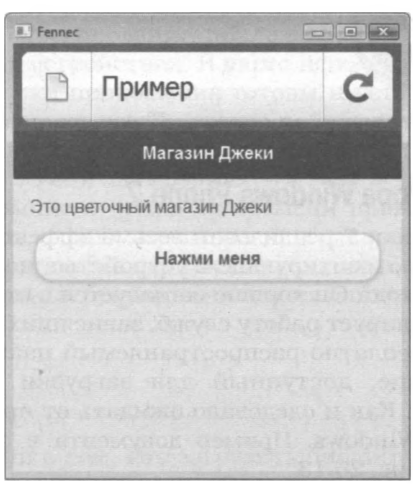


Рис. 26.11. Пример документа в эмуляторе Firefox Mobile

Использование программы просмотра Multi-Browser Viewer

Из платных средств я пользуюсь программой просмотра Multi-Browser Viewer (`http://www.multibrowserviewer.com`). Этот продукт включает в себя более 49 брау-

зеров для тестирования совместимости, хотя большинство из них не являются мобильными и в их число не входят эмуляторы Opera Mobile и Firefox Mobile. Однако он содержит эмуляторы iPhone и iPad, которые используют движок для вывода вебстраниц Apple Safari, что позволяет получить хорошее (хотя и не полное) представление о том, как будет выглядеть страница в этих устройствах. В целом это хороший продукт, и я с успехом использовал его в нескольких проектах, но это, скорее, средство тестирования общего назначения, в котором поддержка мобильных устройств в действительности составляет лишь часть того, что оно может делать. Пример документа в окне эмулятора iPhone представлен на рис. 26.12.

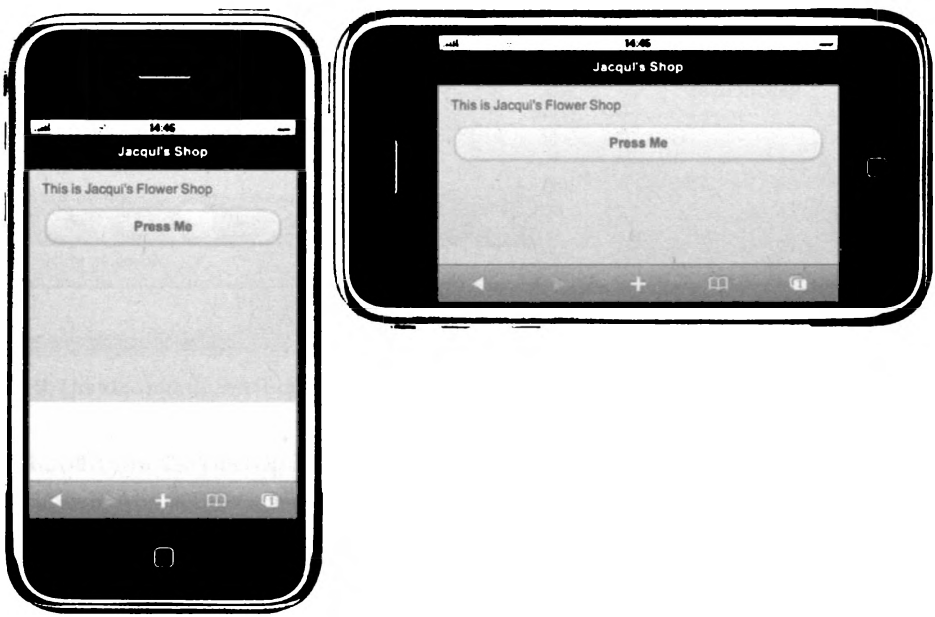


Рис. 26.12. Пример документа в эмуляторе iPhone программы Multi-Browser Viewer

Использование эмулятора Windows Phone 7

Эмулятор Windows Phone 7 реализован весьма эффективно. Это еще один пример эмулятора, полностью имитирующего устройство, но он работает намного быстрее, чем эмулятор Android. Он хорошо согласуется с мобильными устройствами Windows 7 и удачно моделирует работу служб, зависящих от местонахождения устройства. Он входит в бесплатно распространяемый пакет Visual Studio 2010 Express for Windows Phone, доступный для загрузки по адресу `http://www.microsoft.com/express`. Как и следовало ожидать от продукта Microsoft, он поддерживается только в Windows. Пример документа в окне эмулятора Windows Phone 7 представлен на рис. 26.13.

Размер загрузочного файла превышает 500 Мбайт, что далеко от идеала. Но основной проблемой данного эмулятора является то, что внедрение Windows Phone 7 идет очень медленно. Предлагаемые на рынке устройства Windows Phone 7 не пользуются популярностью, и на момент написания этих строк будущее платформы представляется весьма туманным. Обычно я использую этот эмулятор в тех случаях, когда сталкиваюсь с необычным поведением своего приложения и хочу проверить, не связано ли это с особенностями реализации других эмуляторов.

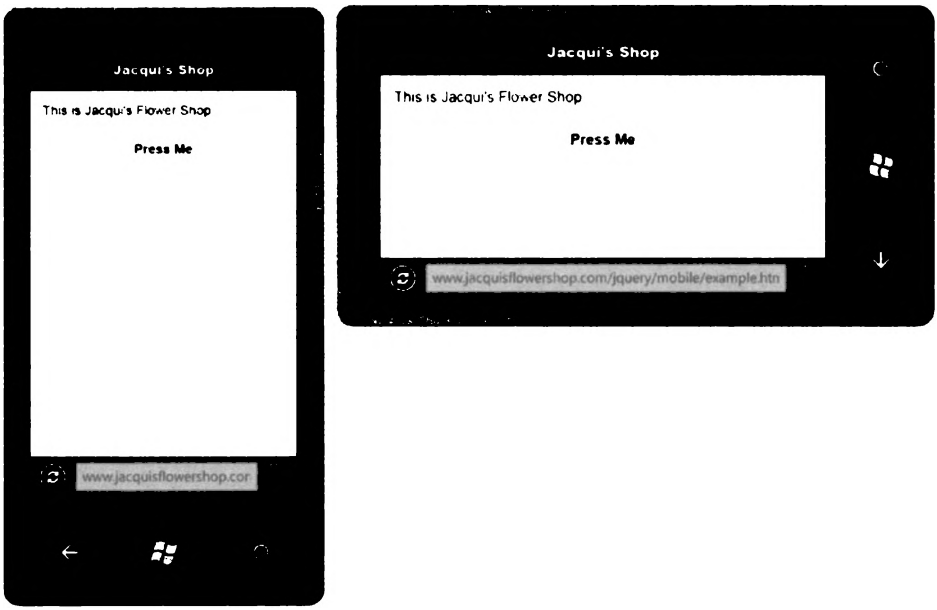


Рис. 26.13. Пример документа в эмуляторе Windows Phone 7

Использование настольных браузеров

В качестве альтернативы эмуляторам Android и iOS я часто пользуюсь настольными браузерами Google Chrome и Apple Safari. Очевидно, что это не то же самое, что мобильные версии этих браузеров, но они имеют общую основу и их удобно использовать для "грубого" тестирования приложений до того, как они будут устанавливаться на реальных устройствах. Я часто использую эти браузеры, когда основные строительные блоки приложения готовы и остается наполнить их функциональной начинкой. Настольные браузеры работают быстрее и более надежны по сравнению с эмуляторами мобильных браузеров (не говоря уже о том, что они работают на имеющихся у меня платформах) и к тому же обладают великолепными средствами отладки. Обычно я время от времени проверяю промежуточные результаты разработки с помощью эмулятора или реального устройства, дабы убедиться в том, что я продвигаюсь в верном направлении, но на ранних стадиях мобильных проектов настольные браузеры являются для меня прекрасным средством разработки.

Резюме

В этой главе вы узнали о том, как загрузить библиотеку jQuery Mobile и включить ее в HTML-документ, а также о сути базового подхода, используемого в jQuery Mobile для автоматического улучшения HTML-документов и отделения страниц от этих документов. Здесь были рассмотрены пользовательские события, которые jQuery Mobile предоставляет разработчикам для упрощения процесса создания приложений, ориентированных на сенсорный интерфейс, а также даны рекомендации общего характера, касающиеся разработки и тестирования мобильных приложений.

ГЛАВА 27

Страницы и навигация

В этой главе описан один из ключевых строительных блоков jQuery Mobile — *страницы*. О них уже упоминалось в главе 26, а здесь будет подробно рассказано и показано, как определяются и настраиваются страницы и как осуществляются переходы между ними. Перечень тем, рассматриваемых в данной главе, приведен в табл. 27.1.

Таблица 27.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Определение страницы jQuery Mobile	Примените к элементу атрибут `data-role` со значением `page`	1
Добавление верхнего и нижнего колонтитулов в страницу	Примените к элементам атрибут `data-role` со значением `header` или `footer`	2, 3
Определение нескольких страниц в документе	Создайте несколько элементов, атрибуты `data-role` которых имеют значение `page`	4
Навигация между страницами	Создайте элемент `a`, элемент `href` которого является атрибутом `id` элемента страницы	5
Определение эффекта перехода для элемента `a`	Примените атрибут `data-transition`	6
Установка глобального эффекта перехода	Присвойте значение параметру `defaultPageTransition`	7
Установка ссылки на страницу в другом документе	Укажите URL-адрес документа в качестве значения `href` элемента `a`	8, 9
Отключение Ajax для одиночной ссылки	Установите для атрибута `data-ajax` значение `false`	10
Глобальное отключение Ajax	Установите для параметра `ajaxEnable` значение `false`	11
Предварительная загрузка страницы	Используйте атрибут `data-prefetch`	12, 13
Изменение текущей страницы	Используйте метод `changePage()`	14
Управление направлением эффекта перехода	Используйте параметр `reverse` при вызове метода `changePage()`	15
Определение времени задержки, по истечении которого отображается диалоговое окно загрузки	Используйте параметр `loadMsgDelay`	16
Отключение диалогового окна загрузки	Используйте параметр `showLoadMsg`	17
Определение текущей страницы	Используйте свойство `activePage`	18
Фоновая загрузка страницы	Используйте метод `loadPage()`	19

Окончание табл. 27.1

Задача	Решение	Листинг
Ответная реакция на загрузку страницы	Используйте события загрузки страниц	20
Ответная реакция на выполнение переходов между страницами	Используйте события переходов между страницами	21

Страницы jQuery Mobile

В главе 26 было продемонстрировано, как определить страницы jQuery Mobile в HTML-документе, используя элементы со специфическими ролями, определяемыми атрибутом `data-role`. Чтобы напомнить вам об этом, в листинге 27.1 воспроизведен пример простой страницы.

Листинг 27.1. Пример простой страницы jQuery Mobile в HTML-документе

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div data-role="page">
       <div data-role="content">
           Это цветочный магазин Джеки
       </div>
    </div>
</body>
</html>
```

Это минимальная страница, содержащая два ключевых элемента, каждый из которых имеет атрибут `data-role`. Элемент, выступающий в качестве страницы (`page`), обозначает область HTML-содержимого, которой соответствует страница jQuery Mobile. Как уже подчеркивалось в главе 26, одной из ключевых характеристик jQuery Mobile является то, что страницы, отображаемые для пользователя, не связаны непосредственно с HTML-элементами, в которых они содержатся.

Другому важному элементу отведена роль `content`. Этот элемент обозначает секцию страницы jQuery Mobile, в которой находится ее содержимое. Страница может содержать несколько секций, лишь одна из которых, как вскоре будет показано, является содержимым. Вид страницы в окне браузера представлен на рис. 27.1.

Добавление верхних и нижних колонтитулов на страницу

В дополнение к секции содержимого страница jQuery Mobile может включать верхний (header) и нижний (footer) колонтитулы. Эти секции обозначаются элементами,

атрибуты `data-role` которых имеют значения соответственно `header` и `footer`. Добавление обеих секций в пример страницы продемонстрировано в листинге 27.2.

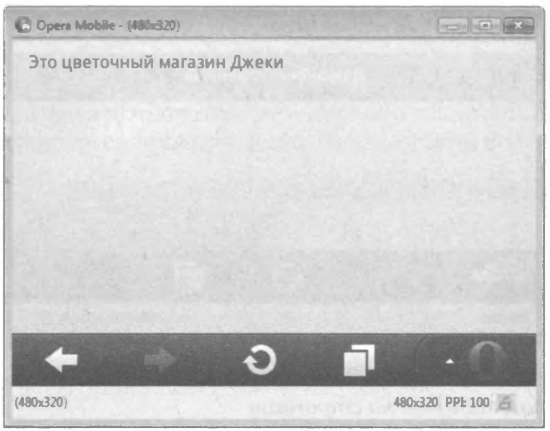


Рис. 27.1. Минимальная страница jQuery Mobile в окне браузера

Листинг 27.2. Добавление верхнего и нижнего колонтитулов на страницу

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="header"
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
        </div>
        <div data-role="footer">
            <h1>Домашняя страница</h1>
        </div>
    </div>
</body>
</html>
```

Результат включения колонтитулов представлен на рис. 27.2. Проблемой верхних и нижних колонтитулов является то, что на небольших экранах они могут занимать значительную часть страницы, как видно на рисунке.

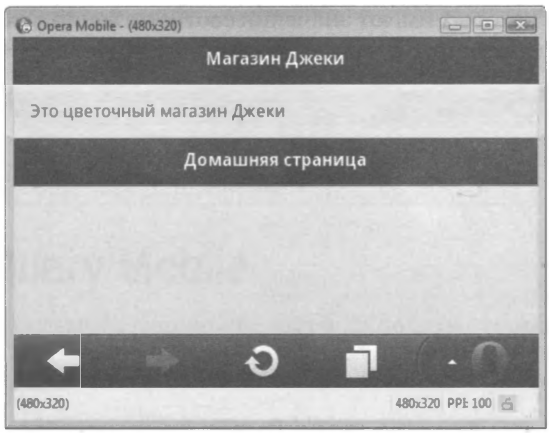


Рис. 27.2. Добавление верхнего и нижнего колонтитулов на страницу

Вы, должно быть, заметили, что нижний колонтитул отображается в конце секции содержимого, а не в нижней части страницы. Возможно, у вас возникнет соблазн принудительно задать позицию нижнего колонтитула с помощью стилей CSS, но браузеры не всегда будут реагировать на это так, как вы рассчитываете. Пример такой попытки скорректировать положение колонтитула представлен в листинге 27.3.

Листинг 27.3. Попытка скорректировать позицию нижнего колонтитула на странице с помощью стилей CSS

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #footer {position: absolute; bottom: 0}
    </style>
</head>
<body>
    <div data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Цветочный магазин Джеки
        </div>
        <div id="footer" data-role="footer">
            <h1>Домашняя страница</h1>
        </div>
```

```
    </div>
</body>
</html>
```

Здесь налицо проблема, связанная с непоследовательной обработкой высоты документа мобильными браузерами. На рис. 27.3 показано, к чему мы приходим в случае браузера Opera. Колонтитул отображается где-то за пределами страницы. На рисунке первый снимок представляет первоначальный вид страницы, а второй — вид страницы после ее прокрутки для отображения колонтитула.

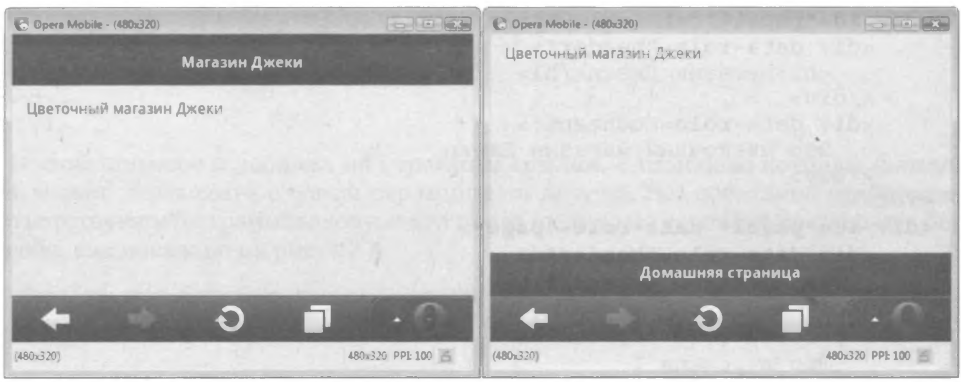


Рис. 27.3. Нижний колонтитул страницы в окне браузера Opera Mobile

Возможно, вы захотите решить проблему путем изменения элемента `meta`, с помощью которого устанавливается размер окна просмотра (viewport). Это ничего не даст. На момент написания книги поддержка высоты окна просмотра в мобильных браузерах отсутствовала. Более того, такой подход рискован. Нельзя заранее сказать, какая часть экрана будет контролироваться браузером. Отсюда следует, что решить проблему, устанавливая высоту окна просмотра в соответствии с размерами устройства, не удастся, а значит, единственное, что остается делать, — задать конкретное значение высоты в пикселях. Но если учесть, что экранное разрешение смартфонов и планшетов, предлагаемых сегодня на рынке, варьируется в широких пределах, то добиться нужного результата этим способом также практически нереально.

Совет. Исходя из этих соображений, я предпочитаю вообще не использовать нижние колонтитулы. Что касается верхних колонтитулов, то я их использую, поскольку они позволяют подчеркивать существование связи между страницами моих приложений jQuery Mobile.

Добавление страниц в документ

В одном документе можно определить несколько страниц jQuery Mobile. В случае простых веб-страниц полезность такого подхода состоит в том, что он позволяет объединить все, что необходимо, в единственном HTML-файле. Пример многостраничного документа приведен в листинге 27.4.

Листинг 27.4. Определение нескольких страниц jQuery Mobile в одном HTML-документе

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
        </div>
   </div>
</body>
</html>
```

В этом примере в документе определены две страницы. Каждой странице с помощью атрибута `id` присвоен уникальный идентификатор, и на основе значений этих идентификаторов осуществляется навигация между страницами. Возможность выполнения переходов между страницами обеспечивается элементами `a`, значения атрибутов `href` которых совпадают с соответствующими значениями атрибутов `id` целевых страниц, как показано на рис. 27.5.

Листинг 27.5. Навигация между страницами

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
        <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
```

```
            Это цветочный магазин Джеки
            <p><a href="#page2">Перейти на страницу 2</a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
    </div>
</body>
</html>
```

В этом примере я добавил на страницы ссылки, с помощью которых пользователь может переходить с одной страницы на другую. Все заботы об отображении соответствующей страницы документа после щелчка на ссылке jQuery Mobile берет на себя, как показано на рис. 27.4.

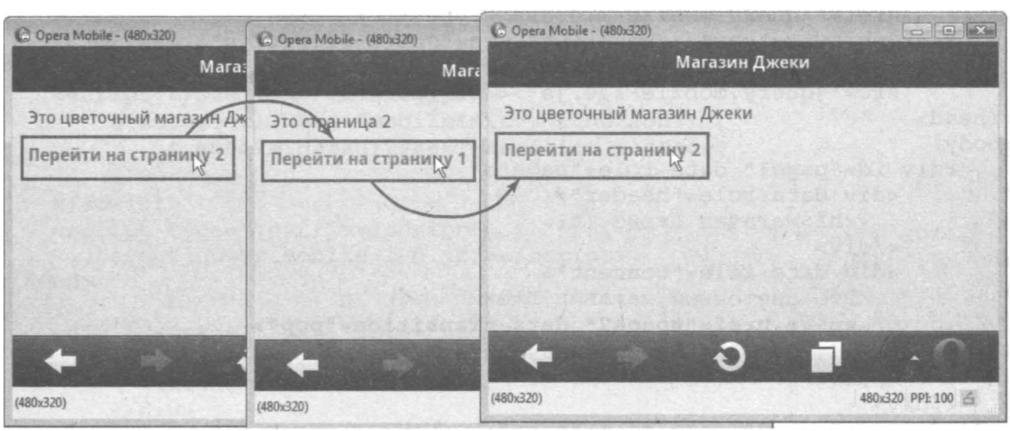


Рис. 27.4. Навигация между страницами документа

Настройка переходов между страницами

Когда пользователь осуществляет навигацию между страницами, jQuery Mobile использует при переходе от одной страницы к другой эффекты анимации. По умолчанию используется эффект `slide`, при котором уходящая страница скользит влево, а новая "наезжает" на нее справа. В jQuery Mobile определены следующие шесть эффектов:

- `slide`;
- `pop`;
- `slideup`;
- `slidedown`;
- `fade`;
- `flip`.

Совет. Эмуляторы мобильных браузеров не могут обеспечить качественную обработку переходов между страницами и, как правило, просто игнорируют их. В то же время на реальных мобильных устройствах все работает так, как надо. Если вы хотите проследить за переходами на настольном компьютере, используйте браузеры Google Chrome и Apple Safari, обеспечивающие достаточно высокое качество эффектов.

Тип анимации для отдельной страницы можно изменить с помощью атрибута `data-transition` элемента `a`, задав значение, соответствующее требуемому анимационному переходу. Пример того, как это можно сделать, приведен в листинге 27.6.

Листинг 27.6. Использование атрибута `data-transition`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><a href="#page2" data-transition="pop">
                Перейти на страницу 2</a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
   </div>
</body>
</html>
```

При щелчке на ссылке, выделенной в листинге полужирным шрифтом, для отображения целевой страницы будет использоваться анимационный переход типа `pop`. Этот эффект будет применяться исключительно к данной ссылке. Для других ссылок на этой или на других страницах того же документа по-прежнему будет использоваться тип анимации, установленный по умолчанию. Если вы хотите вообще отключить эффект анимации при переходах между страницами, установите для атрибута `data-transition` значение `none`.

Совет. Направление воспроизведения анимационного эффекта можно изменить, назначив элементу a атрибут `data-direction` со значением `reverse`. Далее приводится пример обращения перехода и объясняется, какую это может принести пользу.

Для того чтобы изменить анимационный эффект, применяемый ко всем навигационным переходам, нужно установить глобальную опцию. В jQuery Mobile определен параметр `defaultPageTransition`, значение которого можно установить при наступлении события `mobileinit`. Как это можно сделать, показано в листинге 27.7.

Листинг 27.7. Изменение типа анимации, используемого по умолчанию при переходах между страницами

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,

        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("mobileinit", function() {
            $.mobile.defaultPageTransition = "fade"
        })
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><a href="#page2">Перейти на страницу 2</a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
   </div>
</body>
</html>
```

Удобного метода регистрации обработчика события `mobileinit` не существует, поэтому нужно выбрать объект `document` и использовать метод `bind()`. Аргументами этого метода являются имя события, которое вы хотите обрабатывать, и функция-обработчик, которая будет выполняться при наступлении данного события.

Предупреждение. Событие `mobileinit` происходит, как только загружается библиотека сценариев jQuery Mobile, а это означает, что функция-обработчик, предназначенная для изменения глобального параметра jQuery Mobile, должна быть зарегистрирована еще до того, как в элементе `script` встретится ссылка на библиотеку jQuery Mobile. Посмотрите, как это сделано в листинге. Если не поместить вызов метода `bind()` в документе до элемента `script`, который загружает код jQuery Mobile, то функция-обработчик никогда не будет выполнена.

Чтобы изменить значение глобального параметра, необходимо присвоить новое значение свойству объекта `$.mobile`. Поскольку мы хотим изменить значение параметра `defaultPageTransition`, то новое значение следует присвоить свойству `$.mobile.defaultPageTransition`.

```
$.mobile.defaultPageTransition = "fade"
```

Эта инструкция устанавливает `fade` в качестве эффекта по умолчанию. В случае необходимости это значение по-прежнему можно изменить с помощью атрибута `data-transition`, но значением по умолчанию уже не будет являться `slide`.

Связывание с внешними страницами

Вы не обязаны включать все страницы в один документ. Для организации связи с другими страницами можно добавлять ссылки, как в случае использования обычной HTML-разметки. Чтобы продемонстрировать это, был создан файл под названием `document2.html`, содержимое которого представлено в листинге 27.8.

Листинг 27.8. Содержимое файла `document2.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Документ 2</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 1 document2.html
            <p><a href="#page2">Перейти на страницу 2 этого
                документа</a></p>
            <p><a href="example.html">Вернуться к example.html
                </a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
```

```
        <div data-role="content">
            Это страница 2 document2.html
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
    </div>
</body>
</html>
```

Этот документ содержит две страницы jQuery Mobile, сохраняя структуру, знакомую нам по предыдущим примерам. Связь со страницами других документов организуется очень просто. Все, что для этого требуется, — определить элемент a и назначить ему атрибут `href`, указав в нем URL-адрес целевого документа, как показано в листинге 27.9.

Листинг 27.9. Переход на страницу, содержащуюся в другом HTML-документе

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><a href="#page2">Перейти на страницу 2</a></p>
            <p><a href="document2.html">Перейти к document2.html
                </a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
    </div>
</body>
</html>
```

Используя Ajax, jQuery Mobile загружает указанный документ и автоматически отображает его первую страницу с применением эффекта перехода, если таковой был определен. Результат приведен на рис. 27.5.

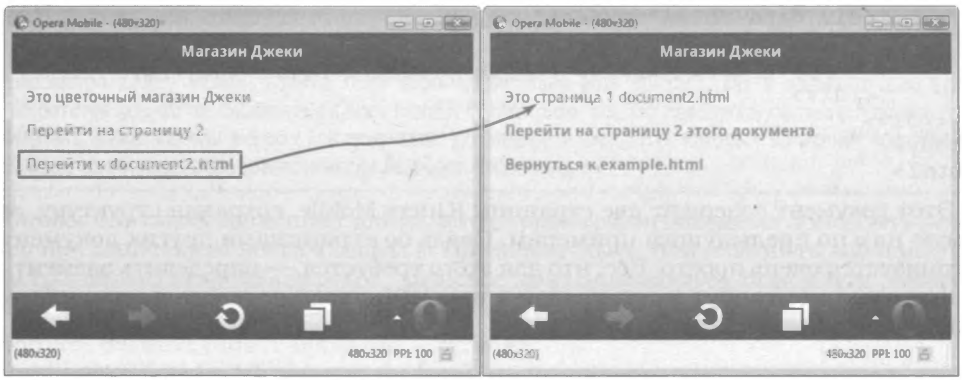


Рис. 27.5. Навигация к странице другого документа

Совет. Библиотека jQuery Mobile автоматически применяет свои стили и улучшает страницы дистанционных документов, загружаемые посредством Ajax-запросов. Это означает, что такие файлы, как `document2.html`, которые используются первичным документом, не требуют подключения библиотек jQuery и jQuery Mobile с помощью элементов `script` и `link`. И все же я рекомендую включать эти ссылки в документы, поскольку при выполнении подобных запросов использование библиотекой jQuery Mobile средств Ajax может быть отменено, и в этом случае автоматическая обработка содержимого выполняться не будет.

Проблемы с идентификацией страниц при выполнении запросов Ajax

К сожалению, работе со ссылками на страницы jQuery Mobile, принадлежащие другим документам, присущи некоторые нюансы. При обработке содержимого, полученного с помощью Ajax-запросов, нормальная работа ссылок на страницы jQuery Mobile может нарушаться. Дело в том, что способ обработки содержимого, получаемого с помощью запросов Ajax, вступает в противоречие со способом определения страниц jQuery Mobile, хотя оба они основаны на использовании атрибутов `id` элементов. Суть проблемы можно увидеть на рис. 27.6.

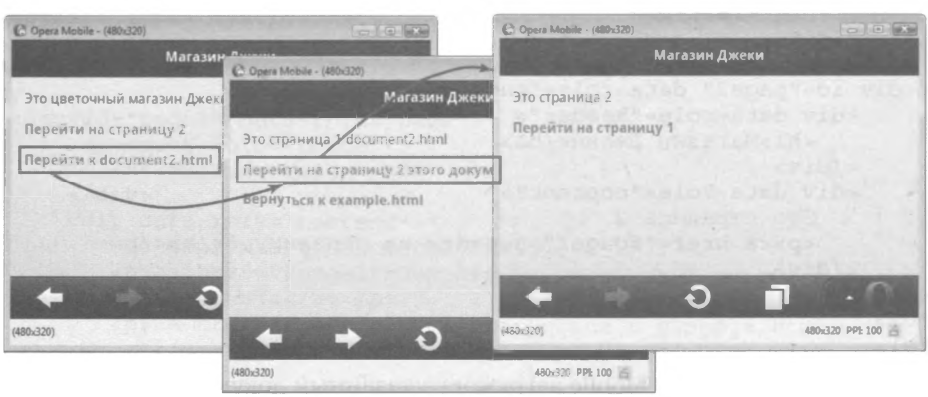


Рис. 27.6. Проблема с обработкой многостраничных документов при использовании Ajax

На этом рисунке изображена последовательность событий, начинающаяся со щелчка на ссылке в документе example.com, приводящего к загрузке документа document2.html. Далее выполнен щелчок на ссылке, который должен был привести к отображению элемента page2 документа document2.html, но результат оказывается неожиданным — на экране действительно отображается элемент page2, только не тот, который предполагался, а тот, который принадлежит документу example.html.

С описанной проблемой можно справиться двумя способами. Способ, которого придерживаюсь я, заключается в том, чтобы определять в каждом HTML-документе лишь одну страницу jQuery Mobile. Именно этот способ рекомендован к использованию командой разработчиков jQuery Mobile.

Второй способ заключается в отказе от использования Ajax при загрузке многостраничных документов. Это позволяет решить указанную проблему со ссылками, но одновременно лишает вас возможности применять эффекты перехода при отображении новой страницы. Чтобы отключить функциональность Ajax для одного элемента a, установите для его атрибута data-ajax значение false, как показано в листинге 27.10.

Листинг 27.10. Отключение функциональности Ajax для одной ссылки

```
<!DOCTYPE html>

<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
           Это цветочный магазин Джеки
           <p><a href="#page2">Перейти на страницу 2</a></p>
           <p><a href="document2.html" data-ajax="false">
              Перейти к document2.html</a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
           Это страница 2
           <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
   </div>
</body>
</html>
```

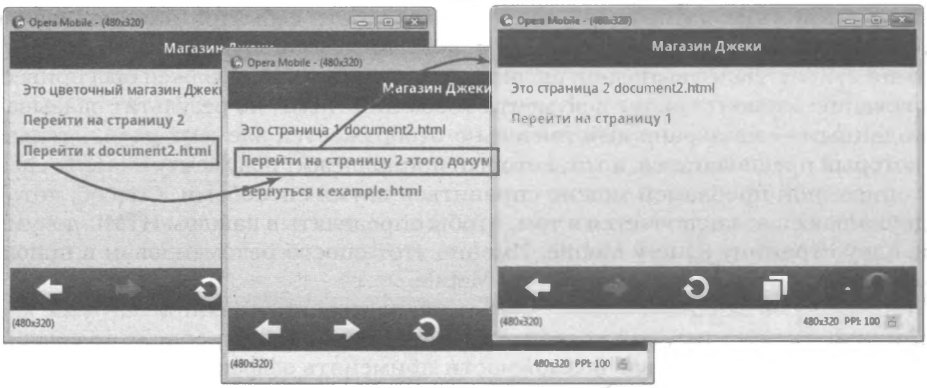


Рис. 27.7. Отключение Ajax во избежание конфликта между идентификаторами элементов

В этом примере я отключил Ajax для ссылки, с помощью которой осуществляется переход к странице `document2.html`. Как показано на рис. 27.7, теперь последовательность межстраничных переходов соответствует ожидаемой.

Чтобы обеспечить отключение функциональности Ajax по умолчанию, используйте глобальный параметр `ajaxEnabled`, как показано в листинге 27.11. Если для этого параметра установлено значение `false`, то при навигации между страницами Ajax использоваться не будет, если только вы не примените к элементу атрибут `data-ajax` со значением `true`.

Листинг 27.11. Отключение Ajax с помощью глобального параметра

```
<!DOCTYPE html>

<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("mobileinit", function() {
            $.mobile.ajaxEnable = false
        })
    </script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><a href="#page2">Перейти на страницу 2</a></p>
            <p><a href="document2.html">
                Перейти к document2.html</a></p>
```

```
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
   </div>
</body>
</html>
```

Предварительная загрузка страниц

Функциональность jQuery Mobile обеспечивает возможность предварительной загрузки документов, в результате чего страницы становятся немедленно доступными для пользователя после щелчка на соответствующей ссылке. Преимуществом такого подхода является то, что приложение начинает быстрее реагировать на запросы пользователя, но при этом возможна загрузка содержимого, к которому пользователь ни разу не обратится. Чтобы продемонстрировать применение описанной возможности, я создал документ `singlepage.html`, содержимое которого представлено в листинге 27.12[1].

Листинг 27.12. Содержимое файла `singlepage.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Единственная страница</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это единственная страница в данном документе
            <button>Привет</button>
            <p><a href="example.html">Вернуться к example.html
                </a></p>
        </div>
    </div>
</body>
</html>
```

[1] Чтобы избежать проблем с отображением кириллицы, сохраните этот файл в кодировке UTF-8. — *Примеч. ред.*

Чтобы разрешить предварительную загрузку страниц, следует назначить элементу a атрибут `data-prefetch`, как показано в листинге 27.13.

Принятие решения о предварительной загрузке содержимого

Принять решение о том, необходима ли предварительная загрузка содержимого, непросто. С точки зрения приложения предварительная загрузка чрезвычайно полезна, поскольку обеспечивает немедленный отклик в ответ на действия пользователя при навигации по страницам документа. Этот фактор становится особенно важным в случае медленных мобильных соединений и сетей с недостаточно хорошим покрытием. Пользователи не любят долго ждать, и если соединение постоянно прерывается, то отсутствие содержимого в нужный для пользователя момент сделает приложение практически бесполезным.

С другой стороны, существует риск того, что содержимое, загруженное на основании прогнозирования возможных переходов пользователя на другие страницы, окажется невостребованным, поскольку пользователь может вообще не перейти на соответствующую страницу. Это может быть крайне нежелательным, если в соответствии с тарифными планами мобильного оператора загрузка данных обходится дорого и может выполняться лишь в пределах ограниченного месячного объема трафика. Применяя предварительную загрузку данных, вы предполагаете, что пользователь считает ваше приложение достаточно ценным, чтобы пожертвовать ради его выполнения частью полосы пропускания (а значит, и денег), а так бывает далеко не всегда. Как это ни прискорбно, но ваше приложение, разработкой которого вы жили более года, может восприниматься пользователем всего лишь как один из множества равноценных для него инструментов.

Я рекомендую не использовать предварительную загрузку страниц. Тем пользователям, которые считают ваше приложение *действительно* ценным, можно позволить самостоятельно принимать решение относительно этого с помощью соответствующей опции.

Листинг 27.13. Предварительная загрузка содержимого

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это цветочный магазин Джеки
            <p><a href="#page2">Перейти на страницу 2</a></p>
            <p><a href="singlepage.html" data-prefetch>
                Перейти к singlepage.html</a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
```

```
        </div>
        <div data-role="content">
            Это страница 2
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
    </div>
</body>
</html>
```

В этом примере jQuery Mobile, в соответствии с нашими инструкциями, осуществляет предварительную загрузку содержимого, указанного с помощью URL-адреса его источника. После щелчка на ссылке переход jQuery Mobile к затребованному содержимому выполняется без какой-либо задержки.

Совет. Если разработка выполняется с использованием быстрой и надежной сети, то проверка целесообразности использования таких средств, как предварительная загрузка содержимого, может оказаться затруднительной. Для подобного рода тестирования я люблю использовать отладочный прокси-сервер HTTP, который отображает для меня отправляемые браузером запросы. Если вы пользователь системы Windows, могу порекомендовать отличное средство Fiddler, имеющее бесчисленное множество настроек (`www.fiddler2.com`).

Использование сценариев для управления страницами jQuery Mobile

Щелчки, выполняемые пользователем на ссылках, не являются единственным средством навигации по страницам. К счастью, jQuery Mobile предоставляет методы и настройки, позволяющие управлять навигацией с помощью JavaScript. В следующих разделах будет продемонстрировано, как воспользоваться преимуществами этих методов, обеспечивающих широкие возможности управления навигацией в веб-приложениях jQuery Mobile.

Изменение текущей страницы

Метод `changePage()` позволяет изменить страницу, отображаемую jQuery Mobile. Пример элементарного использования этого метода, когда смена страницы происходит после щелчка на кнопке, приведен в листинге 27.14.

Листинг 27.14. Изменение страницы, отображаемой jQuery Mobile

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $('button').bind("tap", function(e) {
                var target = this.id == "local" ?
```

```
                    "#page2" : "document2.html";
                $.mobile.changePage(target)
            })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <fieldset class="ui-grid-a">
                <div class="ui-block-a"><button id="local">
                    Локальная</button></div>
                <div class="ui-block-b"><button id="remote">
                    Удаленная</button></div>
            </fieldset>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <p><a href="#page1">Перейти на страницу 1</a></p>
        </div>
    </div>
</body>
</html>
```

В этом примере добавлены две кнопки, щелчок на каждой из которых приводит к вызову метода `changePage()`. При создании этого демонстрационного примера были использованы некоторые возможности jQuery Mobile, о которых рассказывается в последующих главах. С помощью метода `bind()` с кнопками связывается событие `tap`, входящее в небольшой набор настраиваемых вспомогательных событий, определенных в jQuery Mobile. Это событие происходит, когда пользователь касается экрана (или щелкает мышью, если устройство не поддерживает касаний). Данное событие описано наряду с другими событиями jQuery Mobile в главе 26.

В качестве кнопок используются стандартные элементы HTML, которые jQuery Mobile автоматически преобразует в кнопки-виджеты. Настройка кнопок jQuery Mobile описана в главе 29. Наконец, вы, наверное, обратили внимание, что в некоторые элементы добавлены классы `ui-grid-a`, `ui-block-a` и `ui-block-b`. Эти классы являются частью инструментария компоновки страниц, предоставляемого jQuery Mobile, о чем рассказывается в главе 28. Несмотря на то что в примере использованы возможности, которые будут описаны лишь в последующих главах, полученный результат, который представлен на рис. 27.8, довольно прост. После щелчка на любой из кнопок вызывается метод `changePage()`, которому в качестве аргумента передается идентификатор (значение атрибута `id`) локальной страницы или URL-адрес другого документа. Это приводит к загрузке содержимого и отображению эффекта перехода в полной аналогии с тем, что происходит при использовании обычных ссылок.

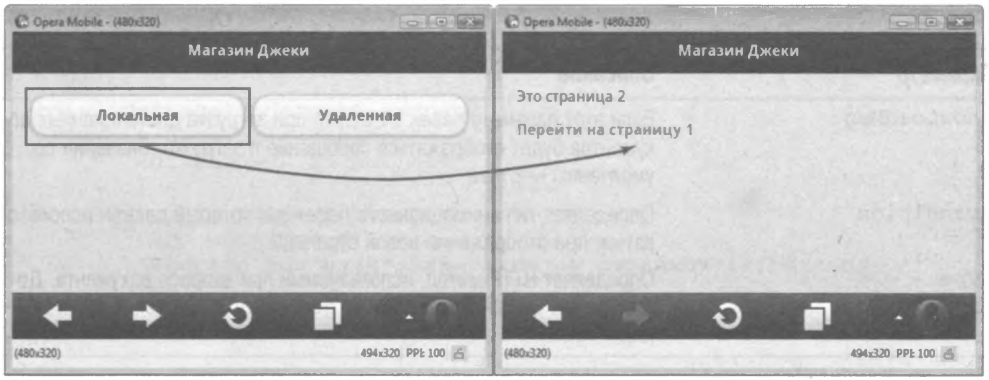


Рис. 27.8. Использование метода `changePage()`

Теперь, после того как вы познакомились с базовым примером использования метода `changePage()`, можно обсудить доступные конфигурационные настройки. Чтобы настроить переходы между страницами, следует передать методу `changePage()` в качестве второго аргумента объект настроек, содержащий значения одного или нескольких параметров. Доступные параметры описаны в табл. 27.2. Для большинства из них лучше оставлять значения, предусмотренные по умолчанию, но в следующих разделах речь пойдет о двух параметрах, значения которых чаще всего нуждаются в изменении.

Таблица 27.2. Параметры метода `changePage()`

Параметр	*Описание*
`allowSamePageTransition`	Если этот параметр равен `false` (значение по умолчанию), то jQuery Mobile будет игнорировать запросы метода `changePage()` в тех случаях, когда целевой является текущая страница
`changeHash`	Если этот параметр равен `true`, то хеш-код в строке URL будет обновлен в соответствии с новым расположением (идентификатор страницы будет включен в URL). Значение по умолчанию — `true`
`data`	Определяет данные, включаемые в Ajax-запрос, который используется для загрузки документа
`dataURL`	Определяет URL-адрес, используемый при обновлении строки URL в браузере. По умолчанию не содержит никакого значения, и в этом случае значение берется из атрибута `id` внутренней страницы или URL-адреса дистанционного документа
`loadMsgDelay`	Определяет длительность периода (в миллисекундах), по истечении которого для пользователя будет отображаться сообщение о загрузке. Значение по умолчанию — `50`
`reloadPage`	Если этот параметр равен `true`, то jQuery Mobile будет перезагружать содержимое, даже если данные уже кешированы. Значение по умолчанию — `false`
`reverse`	Если этот параметр равен `true`, то эффект перехода будет воспроизводиться в обратном направлении. Значение по умолчанию — `false`

Окончание табл. 27.2

Параметр	Описание
showLoadMsg	Если этот параметр равен true, то при загрузке дистанционных документов будет отображаться сообщение о загрузке. Значение по умолчанию — true
transition	Определяет тип анимационного перехода, который должен использоваться при отображении новой страницы
type	Определяет HTTP-метод, используемый при запросе документа. Допустимыми значениями являются get и post. Значение по умолчанию — get

Изменение направления эффекта перехода

К числу наиболее часто используемых мною настроек относится параметр reverse. Все эффекты переходов воспроизводятся jQuery Mobile одинаковым образом, что не всегда подходит для таких, например, ситуаций, когда вы предоставляете пользователю возможность совершить действие, возвращающее его к одной из предыдущих страниц, или реагируете на событие swiperight jQuery Mobile. Суть проблемы проиллюстрирована в листинге 27.15.

Листинг 27.15. Несоответствие между направлением анимационного эффекта и направлением перехода между страницами

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
           $('button').bind("tap", function(e) {
                var target = this.id == "forward" ?
                    "#page2" : "#page1";
                $.mobile.changePage(target, {
                    reverse: (target == "#page1")
                });
           })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
```

```
            Это страница 1
            <button id="forward">Перейти на страницу 2</button>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <button id="back">Вернуться на страницу 1</button>
        </div>
    </div>
</body>
</html>
```

Этот документ содержит две страницы, на каждой из которых имеется кнопка, позволяющая перейти к другой странице. Название кнопки на второй странице — Вернуться на страницу 1. После щелчка на этой кнопке мы меняем направление эффекта перехода на противоположное, присваивая параметру `reverse` значение `true`. В результате этого эффект выглядит намного естественнее, хотя передать это с помощью статического рисунка невозможно. В процессе навигации по документу мы подсознательно настраиваемся на то, что направление "листания" страниц, передаваемое средствами анимации, должно зависеть от того, к какой странице мы переходим — следующей или предыдущей. Смысл этих слов станет совершенно ясен, если вы выполните пример в браузере.

Управление выводом сообщения о загрузке

Если загрузка дистанционного документа средствами Ajax занимает более 50 мс, jQuery Mobile выводит сообщение о загрузке страницы. В случае использования эмулятора мобильного браузера и быстрой сети процесс загрузки документа может происходить настолько быстро, что сообщение о загрузке вообще не успеет появиться. Но если вы используете мобильную сеть передачи данных или, как это сделал я, введете в запрос задержку, то сообщение будет оставаться на экране достаточно долго для того, чтобы его можно было увидеть, как показано на рис. 27.9.

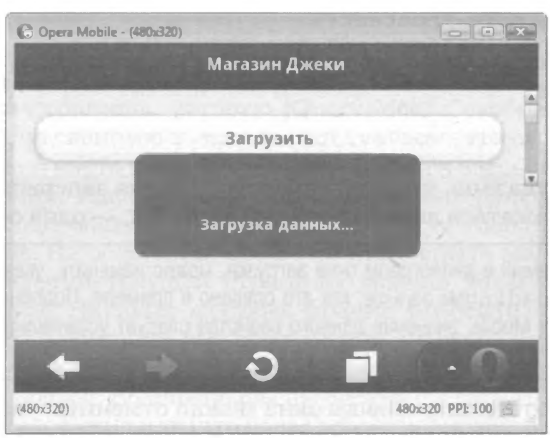


Рис. 27.9. Сообщение о загрузке страницы

Длительность периода задержки, по истечении которого начинает отображаться сообщение, можно изменить, присвоив параметру `loadMsgDelay` нужное значение, как показано в листинге 27.16.

Листинг 27.16. Изменение времени задержки для вывода сообщения о загрузке страницы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">

        $(document).bind("mobileinit", function() {
            $.mobile.loadingMessage = "Загружаются данные..."
        })

        $(document).bind("pageinit", function() {
           $('button').bind("tap", function(e) {
                $.mobile.changePage("document2.html",{
                    loadMsgDelay: 1000
                });
           })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button id="forward">Загрузить</button>
        </div>
    </div>
</body>
</html>
```

В этом примере указано, что длительность периода задержки, по истечении которого может отображаться диалоговое окно загрузки, — одна секунда.

Совет. Текст, отображаемый в диалоговом окне загрузки, можно изменить, указав новое значение глобального свойства `loadingMessage`, как это сделано в примере. Подобно всем другим глобальным свойствам jQuery Mobile, значение данного свойства следует устанавливать в функции, которая выполняется при наступлении события `mobileinit`.

Вывод на экран этого диалогового окна можно отменить, указав при вызове метода `changePage()` значение `false` для параметра `showLoadMsg`. Однако я не рекомендую так поступать, поскольку обратная связь с пользователем никогда не быва-

ет лишней. Тем не менее пример использования данной возможности приведен в листинге 27.17.

Листинг 27.17. Отмена вывода сообщения о загрузке страницы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport"
        content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
           $('button').bind("tap", function(e) {
                $.mobile.changePage("document2.html",{
                    showLoadMsg: false
                });
           })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button id="forward">Загрузить</button>
        </div>
    </div>
</body>
</html>
```

Определение текущей страницы

Для определения страницы, которую jQuery Mobile отображает в данный момент, можно привлечь свойство `$.mobile.activePage`. Это свойство не относится к числу тех, которые я часто использую, и обсуждается здесь исключительно ради полноты рассмотрения. Как правило, мои приложения jQuery Mobile состоят из простых страниц, содержащих небольшие по размеру сценарии и ограниченное количество виджетов. На основании своего личного опыта могу сказать, что необходимость в определении текущей страницы обычно возникает лишь в случае сложных приложений. Никаких жестких правил относительно того, насколько сложным может быть мобильное веб-приложение, не существует, но если вы считаете, что не можете без этого обойтись, рекомендую тщательно проанализировать используемый вами подход. Возможно, вы пытаетесь сделать то, что для мобильного приложения слишком сложно или (что более вероятно) может быть реализовано без привлечения возможностей JavaScript. Пример использования свойства `activePage` приведен в листинге 27.18.

Листинг 27.18. Использование свойства `activePage`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
           $('button').bind("tap", function(e) {
                var nextPages = {
                    page1: "#page2",
                    page2: "#page3",
                    page3: "#page1"
                }
                var currentPageId = $.mobile.activePage
                    .attr("id");
                $.mobile.changePage(nextPages[currentPageId])
           })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 1
            <button id="forward">Выполнить</button>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 2
            <button id="forward">Выполнить</button>
        </div>
    </div>
    <div id="page3" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 3
            <button id="forward">Выполнить</button>
        </div>
    </div>
```

```
</body>
</html>
```

Этот документ содержит три страницы, на каждой из которых имеется кнопка. После щелчка на кнопке считывается свойство `activePage`, позволяющее определить текущую страницу. Свойство `activePage` возвращает объект jQuery, содержащий текущую страницу, поэтому для получения значения атрибута `id` используется метод `attr()` jQuery.

Мой сценарий включает простой объект отображения данных, устанавливающий, какая страница является следующей для каждой из страниц документа, и значение атрибута `id`, полученное из свойства `activePage`, используется в качестве аргумента при вызове метода `changePage()`, что гарантирует прохождение страниц в последовательности, определяемой объектом отображения.

Фоновая загрузка страниц

Метод `loadPage()` позволяет загружать дистанционные документы, не отображая их для пользователя. Эта возможность представляет собой программный эквивалент предварительной загрузки страниц, продемонстрированной ранее. Метод `loadPage()` принимает два аргумента. Первый из них — это URL-адрес загружаемого документа, а второй — необязательный объект настроек. Метод `loadPage()` поддерживает подмножество настроек, используемых методом `changePage()`: `data`, `reloadPage` и `type`. Пример использования метода `loadPage()` приведен в листинге 27.19.

Листинг 27.19. Использование метода `loadPage()`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript"
        src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        var loadedPages = false;

        $(document).bind("pageinit", function() {
            if (!loadedPages) {
                loadedPages = true;
                var pload = $.mobile.loadPage("document2.html")
                pload.done(function() {
                    $('#gobutton').button("enable")
                        .bind("tap", function() {
                        $.mobile.changePage("document2.html")
                    })
                })
            }
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
```

```
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button id="gobutton" disabled>Выполнить</button>
        </div>
    </div>
</body>
</html>
```

В этом примере я использую метод `loadPage()` для предварительной загрузки файла `document2.html`. Метод `loadPage()` возвращает отсроченный объект, который можно использовать для получения уведомлений о загрузке страниц. Об отсроченных объектах мы будем говорить в главе 35, а пока что вам будет достаточно знать лишь то, что для объекта, возвращаемого методом `loadPage()`, можно вызвать метод `done()`, указав в качестве аргумента функцию, которая будет вызываться по завершении Ajax-запроса, выполняемого с помощью метода `loadPage()`.

В данном примере мы используем метод `enable`, вызывая его для виджета кнопки jQuery UI с целью активизации кнопки на странице и регистрации функции-обработчика для события `tap`. Щелчок на кнопке приводит к вызову метода `changePage()`, выполняющего переход к предварительно загруженному документу.

Обратите внимание на глобальную переменную `loadPages`. Событие `page` происходит всякий раз, когда jQuery Mobile инициализирует страницу. Это означает, что оно происходит один раз при загрузке базового документа примера и еще один раз — при загрузке документа `document2.html` посредством Ajax. С помощью переменной `loadPages` мы добиваемся того, чтобы содержимое загружалось только один раз. Ничего страшного не произошло бы и в том случае, если бы метод `callPage()` был вызван дважды (в отсутствие переменной `loadedPages`). В таком случае кешированная копия документа была бы проигнорирована, и вместо этого документ `document2.html` загружался бы повторно. О событиях страниц jQuery Mobile рассказывается в следующем разделе.

Использование событий страниц

В jQuery Mobile определен набор событий, которые можно использовать для получения уведомлений об изменениях, связанных с состоянием страниц. Эти события описаны в табл. 27.3, а наиболее полезные из них подробно обсуждаются в следующих разделах.

Таблица 27.3. События страниц jQuery Mobile

Событие	*Описание*
`pagebeforeload`	Происходит перед выполнением Ajax-запроса на загрузку страницы
`pageload`	Происходит после успешной загрузки страницы посредством Ajax-запроса
`pageloadfailed`	Происходит при неудачном завершении загрузки страницы посредством Ajax-запроса
`pagebeforechange`	Происходит перед переходом на другую страницу
`pagechange`	Происходит после перехода на другую страницу

Окончание табл. 27.3

Событие	Описание
pagechangefailed	Происходит после неудачного перехода на другую страницу (обычно это связано с тем, что запрошенный документ не смог быть загружен)
pagebeforeshow	Происходит перед отображением страницы
pagebeforehide	Происходит перед сокрытием страницы, с которой совершается переход на другую страницу
pageshow	Происходит после отображения страницы
pagehide	Происходит после сокрытия ранее отображавшейся страницы, с которой выполняется переход на другую страницу
pageinit	Происходит по завершении инициализации страницы

Обработка события инициализации страницы

Из всех событий jQuery Mobile событие `pageInit` применяется чаще остальных. Именно на него вы реагируете, если хотите настроить параметры страницы с помощью сценария. Я не буду вновь демонстрировать, как работает это событие, поскольку мы уже неоднократно использовали его в каждом из приводившихся ранее примеров. Можно лишь подчеркнуть, что использование стандартного для jQuery подхода, состоящего в использовании вызова `$(document).ready()`, не может считаться надежным при работе с jQuery Mobile.

Обработка событий загрузки страницы

События `pagebeforeload`, `pageload` и `pageloadfailed` можно использовать для мониторинга Ajax-запросов к дистанционным страницам, как автоматически генерируемым jQuery Mobile, так и программно генерируемым с помощью методов `changePage()` и `loadPage()`. В процессе демонстрации работы метода `loadPage()` для реагирования на загрузку страниц использовался отсроченный объект, однако тот же результат может быть получен с использованием события `pageload` (или, разумеется, с использованием события `pageloadfailed`, если что-то пойдет не так). Предыдущий пример, видоизмененный для использования метода `loadPage()` совместно с событием `pageload`, приведен в листинге 27.20.

Листинг 27.20. Использование события `pageload`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">

        $(document).bind("pageload", function(event, data) {
            if (data.url == "document2.html") {
                $('#gobutton').button("enable")
```

```
                .bind("tap", function() {
                $.mobile.changePage("document2.html")
            })
        }
    })

    var loadedPages = false;
    $(document).bind("pageinit", function() {
        if (!loadedPages) {
            loadedPages = true;
            $.mobile.loadPage("document2.html")
        }
    });
</script>
<script type="text/javascript" src="jquery.mobile-1.0.js">
</script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button id="gobutton" disabled>Выполнить</button>
        </div>
    </div>
</body>
</html>
```

В этом примере я определил функцию, которая будет выполняться при наступлении события `pageload`. Информацию о запросе эта функция получает от jQuery Mobile через объект данных, передаваемый ей в качестве второго аргумента. Для проверки того, что загрузился именно тот объект, который и ожидался, используется свойство `url`. Свойства, определяемые объектом данных для события `pageload`, описаны в табл. 27.4. Как нетрудно заметить, в большинстве своем они соответствуют средствам поддержки Ajax в jQuery, описанным в главах 14 и 15.

Таблица 27.4. Свойства объекта данных события `pageload`

Свойство	*Описание*
`url`	Возвращает URL-адрес, переданный методу `loadPage()` (этот метод используется как в jQuery Mobile при запросе страниц, так и методом `changePage()`)
`absUrl`	Полный URL-адрес
`options`	Параметры Ajax-запроса. Конфигурационные параметры Ajax подробно описаны в главах 14 и 15
`xhr`	Объект jQuery, представляющий Ajax-запрос. Подробное описание этого объекта содержится в главах 14 и 15
`textStatus`	Строка, описывающая состояние запроса. Подробное описание приведено в главах 14 и 15

Реагирование на переходы между страницами

События перехода между страницами можно использовать для получения уведомлений о навигации, осуществляемой пользователем (или программой с помощью метода `changePage()`). Эти события (`pagebeforehide`, `pagehide`, `pagebeforeshow` и

pageshow) происходят каждый раз при смене страницы, даже если страница уже отображалась для пользователя. Пример использования одного из этих событий приведен в листинге 27.21.

Листинг 27.21. Реагирование на сокрытие страницы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        var registeredHandlers = false;
        $(document).bind("pageinit", function() {
            if (!registeredHandlers) {
                registeredHandlers = true;
                $('#page1').bind("pagehide", function(event, data) {
                    $.mobile.changePage($("#page1"))
                })
            }
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
        <a href="document2.html">Перейти к document2.html</a>
        </div>
    </div>
</body>
</html>
```

Совет. Библиотека jQuery Mobile предоставляет дополнительную информацию о событиях, передавая объект данных в качестве второго аргумента функции-обработчику. Этот объект определяет либо свойство `nextPage` (для события `pagebeforehide` и `pagehide`), либо свойство `prevPage` (для событий `pagebeforeshow` и `pageshow`). Данные свойства возвращают объект jQuery, содержащий элемент `page`, который был только что скрыт или показан.

В этом примере мы регистрируем функцию-обработчик для связанного с элементом `page1` события `pagehide`, выбирая нужный элемент и вызывая метод `bind()`. Это означает, что мы будем получать данное событие только в том случае, когда скрывается именно выбранная страница. Это довольно примитивный пример, поскольку при наступлении события `pagehide` метод `changePage()` используется всего лишь для возврата к странице `page1`, но, как бы то ни было, он позволяет продемонстрировать, как работает данное событие. Обратите внимание: мы вновь

используем переменную, с помощью которой достигаем того, что функция-обработчик будет регистрироваться только один раз. Если бы мы этого не сделали, то при загрузке страницы `document2.html` для одного и того же события, привязанного к одному и тому же элементу, были бы зарегистрированы две функции.

Резюме

В этой главе речь шла о страницах — основных строительных блоках jQuery Mobile. Возможность существования нескольких страниц в одном документе — уникальная особенность jQuery Mobile, которая может быть очень полезной в случае небольших и несложных приложений. Однако в более серьезных проектах каждая страница должна содержаться в собственном документе, и исходить следует из того, что jQuery Mobile будет загружать страницы посредством Ajax-запросов.

ГЛАВА 28

Диалоговые окна, темы и макеты

В этой главе описаны три функциональных компонента jQuery Mobile, которые могут пригодиться вам при создании мобильных веб-приложений. Каждый из них сравнительно прост, но все они будут использоваться в последующих главах, и поэтому своевременное ознакомление с ними не будет лишним. Эти компоненты позволяют создавать диалоговые окна, применять темы к страницам и элементам и компоновать элементы с помощью сетки, состоящей из набора столбцов. Ни одна из этих возможностей сама по себе не является чем-то из ряда вон выходящим, но они являются тем инструментарием, который облегчает жизнь разработчикам веб-приложений. Перечень тем, рассматриваемых в данной главе, приведен в табл. 28.1.

Таблица 28.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Отображение страницы в виде диалогового окна	Добавьте атрибут `data-rel` в навигационную ссылку и присвойте ему значение `dialog`	1
Добавление в диалоговое окно кнопки закрытия	Добавьте на страницу диалогового окна ссылку с атрибутом `data-rel` и присвойте ему значение `back`	2
Добавление в диалоговое окно кнопки для перехода на другую страницу	Добавьте на страницу диалогового окна ссылку и установите для ее атрибута `href` значение, равное URL-адресу целевой страницы	3
Программное управление диалоговыми окнами	Используйте метод `changePage()` для открытия диалогового окна и перехода на другую страницу. Используйте метод `dialog("close")` для закрытия диалогового окна	4
Добавление образца стиля в страницу или элемент	Используйте атрибут `data-theme`, значением которого является требуемый образец стиля	5, 6
Компоновка элементов с помощью сетки	Используйте компоновочные CSS-классы jQuery Mobile	7

Создание диалоговых окон

Одним из замечательных свойств страниц jQuery Mobile является то, что с их помощью можно легко создавать диалоговые окна. Это делается путем применения атрибута `data-rel` к элементу `a`, обеспечивающему переход на нужную страницу, и присвоения этому атрибуту значения `dialog`. Соответствующий пример приведен в листинге 28.1.

Листинг 28.1. Использование страницы в качестве диалогового окна

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <a href="#page2" data-rel="dialog">
                Показать диалоговое окно</a>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
           <h1>Вы щелкнули на ссылке!</h1>
        </div>
        <div data-role="content">
            Это область содержимого диалогового окна
        </div>
    </div>
</body>
</html>
```

В этом примере присутствуют две обычные страницы jQuery Mobile. Ссылка на первой странице практически ничем не отличается от ссылок, которые использовались в предыдущих примерах, за исключением того, что ей назначен атрибут `data-rel`, которому присвоено значение `dialog`. Когда пользователь щелкает на ссылке, jQuery Mobile извлекает страницу и представляет ее в виде диалогового окна, как показано на рис. 28.1.

Любопытно отметить, что для получения подобного результата вам не нужно прилагать никаких усилий. Вся настройка заключена в одной-единственной навигационной ссылке и не требует ни одной дополнительной строки JavaScript-кода.

Добавление кнопок в диалоговое окно

Чтобы закрыть базовое диалоговое окно и вернуться на предыдущую страницу, пользователь должен щелкнуть на значке в левом верхнем углу окна. Можно дополнить диалоговое окно кнопкой закрытия путем добавления в него элемента `button` и использования дополнительного атрибута данных, как показано в листинге 28.2.

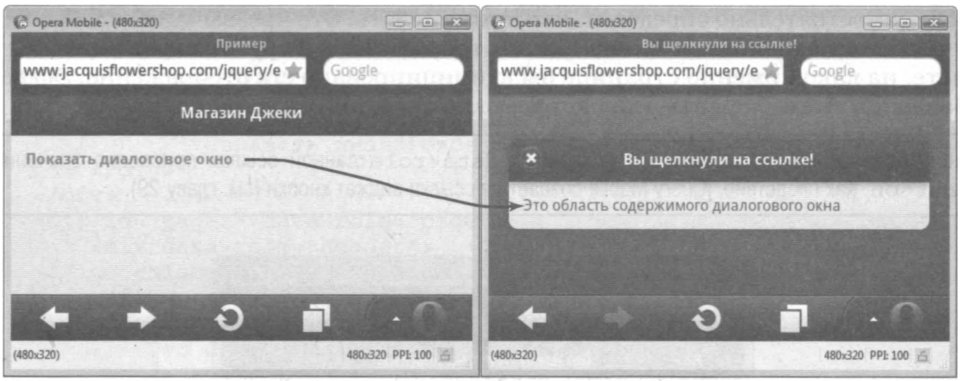


Рис. 28.1. Отображение страницы в виде диалогового окна

Листинг 28.2. Добавление кнопки закрытия диалогового окна

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <a href="#page2" data-rel="dialog">
                Показать диалоговое окно</a>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
           <h1>Вы щелкнули на ссылке!</h1>
        </div>
        <div data-role="content">
            Это область содержимого диалогового окна
            <a href="#" data-role="button" data-rel="back">
                Закрыть</a>
        </div>
    </div>
</body>
</html>
```

Здесь я добавил элемент a и применил к нему атрибут `data-rel`, которому присвоил значение `back`. Указания адресата ссылки не требуется. Достаточно лишь установить символ "решетки" (#) в качестве значения атрибута `href`, и jQuery

Mobile самостоятельно определит, что нужно делать. Это чрезвычайно удобно в тех случаях, когда диалоговое окно должно отображаться с других страниц, но вы не знаете, на какой именно странице было инициировано его отображение. Эффект добавления этого элемента в диалоговое окно представлен на рис. 28.2.

Совет. Должно быть, вы заметили, что атрибуту `data-role` данной ссылки присвоено значение `button`. Как следствие, jQuery Mobile создает из ссылки виджет кнопки (см. главу 29).

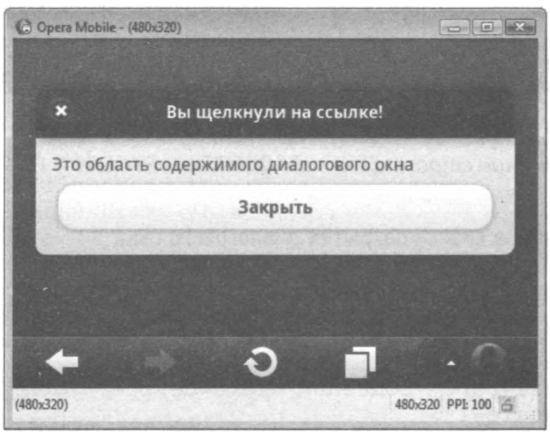


Рис. 28.2. Добавление кнопки закрытия диалогового окна

Добавление другой кнопки в диалоговое окно

В диалоговое окно можно добавлять другие кнопки, функциональность которых вы определяете по своему усмотрению. Если вы хотите предоставить пользователю возможность перехода на другую страницу, то для этого достаточно добавить элемент `a` и указать в его атрибуте `href` URL-адрес требуемой страницы. Именно так и сделано в листинге 28.3, в котором атрибуту `data-role` присваивается то же значение, что и в предыдущем примере, в результате чего jQuery Mobile превращает ссылку в кнопку.

Листинг 28.3. Добавление навигационной ссылки в диалоговое окно

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
```

```
        <div data-role="header">
           <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <a href="#page2" data-rel="dialog">
                Показать диалоговое окно</a>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
           <h1>Вы щелкнули на ссылке!</h1>
        </div>
        <div data-role="content">
            Это область содержимого диалогового окна
            <a href="#page3" data-role="button">OK</a>
            <a href="#" data-role="button" data-rel="back">
                Закрыть</a>
        </div>
    </div>
    <div id="page3" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 3. Вы перешли сюда с помощью диалогового
            окна.
        </div>
    </div>
</body>
</html>
```

В этом примере добавлен элемент a, переносящий пользователя на добавленную в документ страницу 3. Процесс перехода из диалогового окна на страницу 3 проиллюстрирован на рис. 28.3.

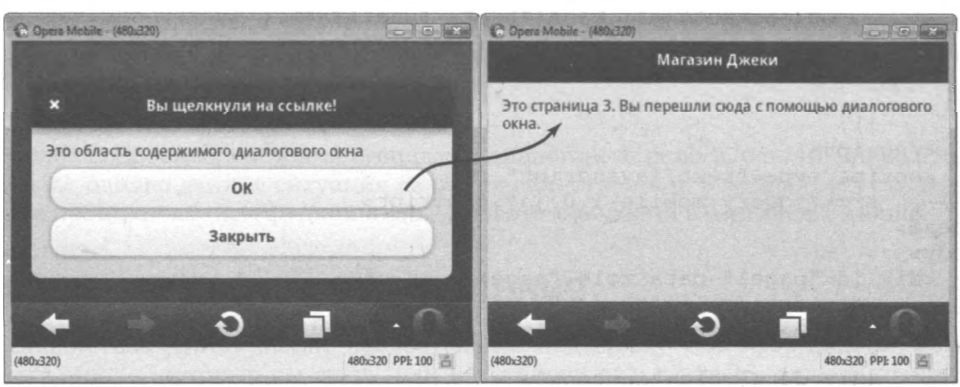


Рис. 28.3. Добавление навигационной ссылки-кнопки в диалоговое окно

Совет. Поскольку jQuery Mobile не включает диалоговое окно в журнал посещенных страниц, после перехода из диалогового окна на другую страницу щелчок на кнопке возврата вернет вас на ту страницу, которая отображалась на экране до диалогового окна.

Управление диалоговым окном из программы

Несмотря на то что jQuery Mobile пытается свести к минимуму объем кода, который должен быть написан вами, иногда возникает необходимость взять на себя непосредственное управление всем, что связано с использованием диалогового окна. Пример открытия и закрытия диалогового окна из программы приведен в листинге 28.4.

Листинг 28.4. Открытие и закрытие диалогового окна из программы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript"
        src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {

            $('#openDialog').bind("tap", function() {
                $.mobile.changePage("#page2", {
                    role: "dialog"
                })
            })

            $('#closeDialog').bind("tap", function() {
                $('#page2').dialog("close")
            })

            $('#okButton').bind("tap", function() {
                $.mobile.changePage("#page3")
            })

        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button id="openDialog">Показать диалоговое
                окно</button>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
           <h1>Вы щелкнули на ссылке!</h1>
        </div>
```

```
        <div data-role="content">
            Это область содержимого диалогового окна
            <button id="okButton">OK</button>
            <button id="closeDialog">Закрыть</button>
        </div>
    </div>
    <div id="page3" data-role="page">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это страница 3. Вы перешли сюда с помощью диалогового
            окна.
        </div>
    </div>
</body>
</html>
```

В этом примере ссылки заменены кнопками, поскольку для щелчка на кнопке, находящейся вне формы, действие по умолчанию не предусмотрено. Я также добавил элемент `script`, в котором зарегистрировал для каждого элемента `button` обработчики события `tap`. (Здесь используются отдельные вызовы метода `bind()`, поскольку это упрощает объяснение примера, но ничто не мешает объединить все три функции в одну, как это делалось в предыдущих частях.)

Чтобы открыть диалоговое окно, достаточно вызвать метод `changePage()`, указав в качестве аргументов URL-адрес требуемой страницы и объект настроек, задающий для свойства `role` значение `dialog`, как показано ниже.

```
$.mobile.changePage("#page2", {
    role: "dialog"
})
```

Метод `changePage()` рассматривался в главе 27. Этот же метод мы используем, когда хотим покинуть диалоговое окно и перейти на другую страницу.

```
$.mobile.changePage("#page3")
```

При закрытии диалогового окна из программы библиотека jQuery Mobile опирается на возможности jQuery UI.

```
$('#page2').dialog("close")
```

Более подробные сведения о виджете диалогового окна jQuery UI приведены в главе 22, однако данная ситуация является единственной, в которой мы сталкиваемся с проникновением функциональности этого виджета в мир jQuery Mobile.

Применение тем оформления

Библиотека jQuery Mobile предоставляет поддержку тем оформления. Файлы jQuery Mobile, которые вы загрузили и установили в главе 26, включают оригинальную тему оформления, применяемую по умолчанию. Наряду с этим вам предоставляется возможность создавать собственные темы с помощью приложения jQuery Mobile ThemeRoller — видоизмененного варианта визуального редактора тем оформления с тем же названием, с которым вы уже сталкивались, создавая пользовательские темы оформления для jQuery UI.

Примечание. Как уже было отмечено в главе 26, подробное рассмотрение вопросов, относящихся к созданию и использованию пользовательских тем оформления, в мои планы не входит. Приложение jQuery Mobile ThemeRoller — сравнительно поздняя добавка в jQuery Mobile, с функционированием которой на момент написания книги были связаны определенные проблемы. Надеюсь, что к тому времени, когда вы будете читать эти строки, указанные проблемы будут разрешены.

Тема оформления jQuery Mobile включает в себя одну или несколько *палитр* (swatches), или *цветовых схем*, которые представляют собой набор стилей, применяемых к различным типам элементов. Тот факт, что мы не будем заниматься созданием пользовательской темы, еще не означает, что вы не можете воспользоваться приложением ThemeRoller для просмотра стандартной темы, используемой по умолчанию. Палитры обозначаются буквами от `A` до `Z`. Стандартная тема включает пять палитр с именами от `A` до `F`. Чтобы просмотреть стандартную тему оформления, перейдите в браузере по адресу `http://jquerymobile.com/themeroller`, щелкните на ссылке Import, а затем скопируйте содержимое файла `jquery.mobile-1.0.css` в буфер обмена и вставьте его в диалоговое окно (это тот самый CSS-файл, который вы загрузили и установили в главе 26). Приложение ThemeRoller обработает CSS-стили и отобразит палитры, используемые по умолчанию, как показано на рис. 28.4.

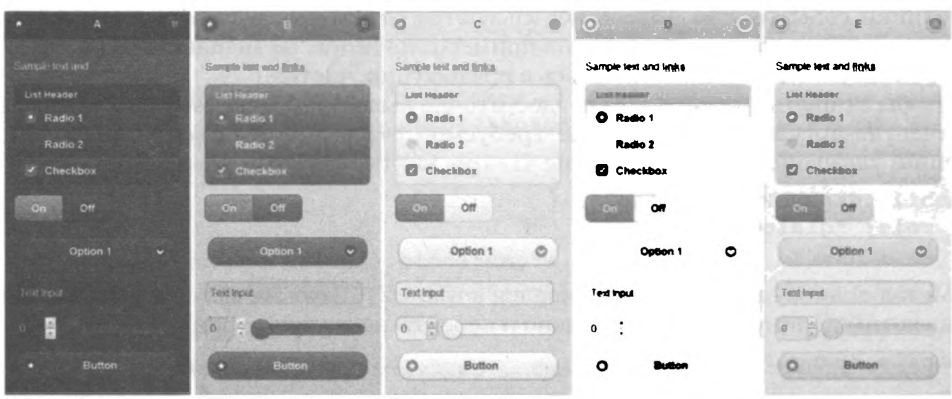


Рис. 28.4. Образцы палитр темы, используемой по умолчанию

Совет. На момент написания книги приложение ThemeRoller не могло правильно выполнять синтаксический анализ темы оформления, используемой по умолчанию, для извлечения CSS-разметки. Попытки модифицировать любой из стилей, входящих в палитру, приводили к генерации дефектных таблиц CSS, и ни одна из палитр не отображалась так, как следовало.

Применение темы оформления к странице jQuery Mobile сводится к использованию атрибута `data-theme` с указанием имени требуемой палитры. Соответствующий пример приведен в листинге 28.5.

Совет. Также существует палитра под названием `active`, используемая для визуального выделения выбранных кнопок. Ее автоматически применяет jQuery Mobile, но допускается и ее непосредственное использование. В последнем случае вы должны быть уверены в том, что не приведете своими действиями к возникновению конфликта между активными элементами, что может сбивать пользователей с толку.

Листинг 28.5. Использование палитр в теме оформления

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="a">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это тема A
            <a href="#page2" data-role="button">
                Переключить тему</a>
        </div>
    </div>
    <div id="page2" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Это тема B
            <a href="#page1" data-role="button">
                Переключить тему</a>
        </div>
    </div>
</body>
</html>
```

Здесь атрибут `data-theme` применен к элементу `page`, что равносильно применению указанной палитры ко всем дочерним элементам, находящимся в элементе `page`. В этом примере специально предусмотрены две страницы, чтобы, с одной стороны, продемонстрировать, как одно и то же содержимое отображается в двух контрастирующих схемах, а с другой стороны, подчеркнуть, что в jQuery Mobile не существует надежного способа изменить тему, коль скоро на страницу распространяется процесс автоматического улучшения. Атрибут `data-theme` преобразуется в набор CSS-классов, которые применяются к элементам на стадии инициализации страницы, и поэтому изменение значения данного атрибута никоим образом не приведет к изменению классов. Как выглядят эти две страницы, показано на рис. 28.5.

Применение палитр к отдельным элементам

В предыдущем примере было продемонстрировано, как изменять стиль целых страниц, но палитры можно применять и поэлементно, смешивая и комбинируя их для получения какого-либо специфического эффекта. Соответствующий пример приведен на рис. 28.6.

Рис. 28.5. Использование разных палитр в двух страницах одного и того же приложения

Листинг 28.6. Применение палитр к отдельным элементам

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="cp1251">
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript"
        src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="a">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <a href="document2.html" data-role="button"
                data-theme="b">Нажми меня</a>
            <a href="document2.html" data-role="button"
                data-theme="c">Нажми меня</a>
            <a href="document2.html" data-role="button"
                data-theme="d">Нажми меня</a>
        </div>
    </div>
</body>
</html>
```

В этом примере атрибут `data-theme` применен к странице и трем элементам `button` с указанием разных палитр для каждого из них. Результат приведен на рис. 28.6.

Предлагаемая в jQuery Mobile поддержка тем оформления отличается простотой, и с ней легко работать. Было бы неплохо иметь возможность изменять палит-

ры, используемые элементами, "на лету", но даже без этого их можно с успехом использовать для настройки внешнего вида мобильных приложений.

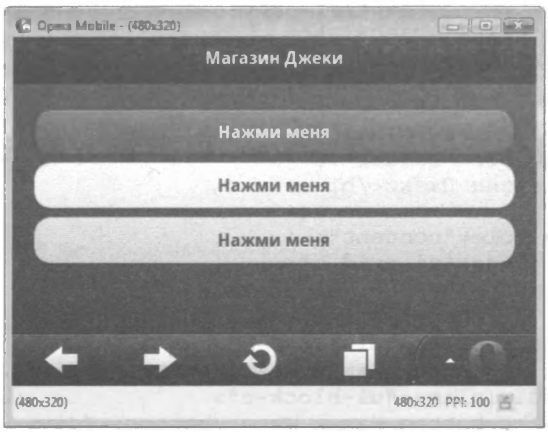


Рис. 28.6. Применение различных палитр к элементам одной и той же страницы

Создание макетных сеток

В jQuery Mobile определены полезные CSS-классы, обеспечивающие размещение содержимого мобильных страниц по ячейкам макетной сетки. Подобные классы вы могли бы создать и сами, но готовые библиотечные классы будут весьма кстати и сократят время разработки, особенно в случае простых мобильных приложений.

В jQuery Mobile определены четыре таких *макетных класса* (layout classes), каждому из которых соответствует разное число столбцов макетной сетки (табл. 28.2).

Таблица 28.2. Классы макетной сетки jQuery Mobile

Класс CSS	*Число столбцов*	*Класс CSS*	*Число столбцов*
`ui-grid-a`	2	`ui-grid-c`	4
`ui-grid-b`	3	`ui-grid-d`	5

Для компоновки содержимого по ячейкам сетки следует применить один из классов `ui-grid` к контейнерному элементу, а к элементам его содержимого — классы `ui-block-a`, `ui-block-b` и так далее для отдельных столбцов. Пример использования указанных классов для создания простой макетной сетки приведен в листинге 28.7.

Листинг 28.7. Компоновка элементов с использованием макетных классов jQuery Mobile

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="cp1251">
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
```

```
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div class="ui-grid-b">
                <div class="ui-block-a">
                    <button>Нажми меня</button></div>
                <div class="ui-block-b">
                    <button>Нажми меня</button></div>
                <div class="ui-block-c">
                    <button>Нажми меня</button></div>
            </div>
            <div class="ui-grid-a">
                <div class="ui-block-a">
                    <button>Нажми меня</button></div>
                <div class="ui-block-b">
                    <button>Нажми меня</button></div>
            </div>
            <div><button>Нажми меня</button></div>
        </div>
    </div>
</body>
</html>
```

В этом примере создаются две макетные сетки — с тремя и двумя столбцами. В каждом столбце содержится элемент `button`. Кроме того, в документ добавлена кнопка, расположенная вне макетной сетки, присутствие которой подчеркивает характер применяемой по умолчанию компоновки, приводящей к растягиванию элементов по всей ширине экрана. Результат представлен на рис. 28.7.

Рис. 28.7. Компоновка элементов с помощью сетки

Резюме

В этой главе вы познакомились с тремя полезными функциональными компонентами jQuery Mobile: диалоговыми окнами, образцами стилей и макетными сетками. Они не относятся к числу наиболее важных, но их стоило описать, поскольку они используются в последующих главах. Несмотря на свою простоту, перечисленные функциональные возможности позволяют формировать структуру большинства приложений jQuery Mobile, и поэтому их знание вам, несомненно, пригодится.

ГЛАВА 29

Кнопки и сворачиваемые блоки

В этой главе описаны два виджета jQuery Mobile: *кнопки* и *сворачиваемые блоки*. Виджет Button jQuery Mobile напоминает виджет Button jQuery UI, с которым вы уже работали в предыдущих главах, с той лишь разницей, что при создании и использовании простых кнопок jQuery Mobile можно вообще обойтись без пользовательского JavaScript-кода. Сворачиваемый блок работает подобно одиночной панели виджета Accordion jQuery UI. В действительности сворачиваемые блоки могут использоваться не только по отдельности, но и в виде группы, аналогичной простому виджету Accordion. Перечень тем, рассматриваемых в данной главе, приведен в табл. 29.1.

Таблица 29.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Автоматическое создание виджета Button (кнопок jQuery Mobile)	Добавьте элемент `button` или элемент `input` с типом `submit`, `reset` или `button`	1
Предотвращение автоматического создания кнопок jQuery Mobile	Примените атрибут `data-role` со значением `none`	2
Создание кнопок из других элементов	Примените атрибут `data-role` со значением `button`	3
Реагирование на события кнопки	Обработайте событие `click` или одно из упрощенных событий jQuery Mobile	4
Добавление значка в кнопку	Используйте атрибуты `data-icon` и `data-iconpos`	5
Предотвращение заполнения виджетом Button всей ширины экрана	Используйте атрибут `data-inline`	6
Создание группы кнопок	Используйте атрибут `data-role` со значением `controlgroup`. Для изменения ориентации группы используйте атрибут `data-type`	7, 8
Создание сворачиваемого блока	Примените атрибут `data-role` со значением `collapsible`. Убедитесь в наличии элемента `header` в качестве первого дочернего элемента	9
Применение палитры к сворачиваемому блоку	Используйте атрибут `data-content-theme`	10

Окончание табл. 29.1

Задача	Решение	Листинг
Установка начального состояния сворачиваемого блока	Используйте атрибут `data-collapsed`	11
Получение уведомлений при свертывании и развертывании сворачиваемого блока	Используйте события `collapse` и `expand`	12
Свертывание и развертывание сворачиваемого блока программным способом	Сгенерируйте событие `collapse` или `expand`	13
Создание виджета Accordion	Используйте атрибут `data-role` со значением `collapsible-set`	14

Использование кнопок jQuery Mobile

Ранее в некоторых примерах мы уже использовали кнопки jQuery Mobile, так что сейчас самое время вернуться к их рассмотрению и объяснить, как они работают.

Автоматическое создание кнопок

Частью процесса автоматического улучшения страниц средствами jQuery Mobile является создание виджетов Button (т.е. кнопок) из элементов `button` или элементов `input`, атрибут `type` которых имеет одно из следующих значений: `submit`, `button` или `image`. Никаких дополнительных действий с вашей стороны не требуется, поскольку всю работу автоматически выполняет jQuery Mobile. Пример страницы, содержащей некоторые из автоматически обрабатываемых элементов, приведен в листинге 29.1.

Листинг 29.1. Автоматическое создание кнопок

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button>Button</button>
            <input type="submit" value="Input (Submit)" />
            <input type="reset" value="Input (Reset)" />
            <input type="button" value="Input (Button)" />
```

```
        </div>
    </div>
</body>
</html>
```

На рис. 29.1 показаны виджеты кнопок, созданные для элементов каждого из вышеперечисленных типов.

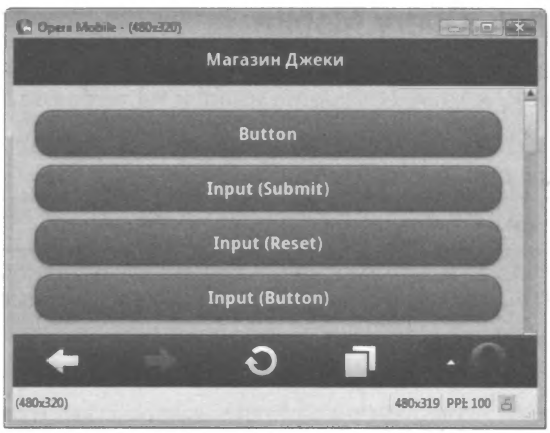


Рис. 29.1. Виджеты кнопок, автоматически созданные jQuery Mobile

Предотвращение автоматического создания кнопок

Чтобы предотвратить автоматическое создание виджетов кнопок jQuery Mobile, назначьте соответствующему элементу атрибут `data-role` со значением `none`, как показано в листинге 29.2.

Листинг 29.2. Предотвращение автоматического создания виджетов кнопок jQuery Mobile

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button>Button</button>
            <input type="submit" value="Input (Submit)" />
            <input type="reset" value="Input (Reset)"
```

```
                data-role="none" />
            <input type="button" value="Input (Button)" />
        </div>
    </div>
</body>
</html>
```

Настройка кнопок jQuery Mobile

В jQuery Mobile определен ряд атрибутов данных, которые можно использовать для настройки кнопок, а также создания кнопок из различных типов элементов. Эти атрибуты описаны в табл. 29.2.

Таблица 29.2. Атрибуты данных для кнопок

Атрибут	*Описание*
`data-corners`	Если этот атрибут равен `true`, то кнопки будут иметь скругленные углы. Значению `false` соответствуют прямые углы. Значение по умолчанию — `true`
`data-icon`	Определяет значок, используемый для кнопки
`data-iconpos`	Определяет позицию значка, если он используется
`data-inline`	Создает кнопку, размер которой определяется размером содержимого (а не шириной экрана)
`data-shadow`	Если этот атрибут равен `true`, то к кнопкам добавляется тень. Значению `false` соответствует отсутствие тени. Значение по умолчанию — `true`

Создание кнопок из других элементов

Библиотека jQuery Mobile также позволяет создавать кнопки из других элементов. В предыдущих главах было показано, как создать виджет кнопки из элемента a путем назначения ему атрибута `data-role` со значением `button`. Точно так же можно поступать и в отношении других элементов, например `div`. Соответствующий пример приведен в листинге 29.3.

Листинг 29.3. Создание кнопок из других элементов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
```

```
        <div data-role="content">
            <a href="document2.html" data-role="button">
                Элемент A</a>
            <div data-role="button">Элемент DIV</div>
        </div>
    </div>
</body>
</html>
```

Результат выполнения этого примера представлен на рис. 29.2.

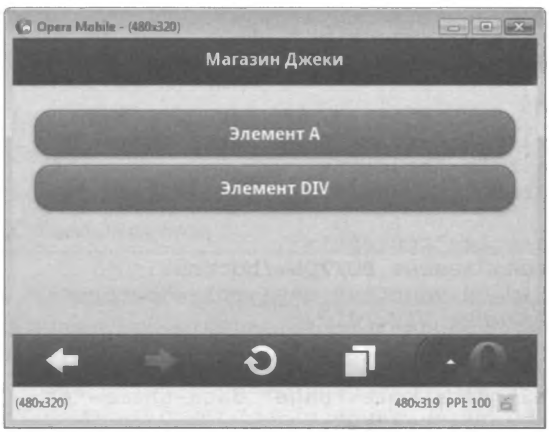


Рис. 29.2. Создание виджетов кнопок с использованием других элементов

Работа с событиями кнопки

Для кнопок, создаваемых из элементов `input` или `a`, предусмотрено стандартное действие, выполняемое по умолчанию после щелчка на кнопке. Действием по умолчанию для элементов `input` является отправка формы, содержащей данный элемент. (Более подробно об использовании форм совместно с jQuery Mobile речь пойдет в главе 30.) Для элемента `a` таким действием является переход к странице, на которую указывает атрибут `href`.

Учитывая сложности работы с событиями касаний, описанными в главе 26, я рекомендую создавать кнопки из элементов `input` и `a` лишь в тех случаях, когда предусмотренное по умолчанию действие вам действительно необходимо. Если же требуется другой результат, используйте для создания виджета кнопки какой-либо другой элемент и организуйте работу с этим элементом наиболее удобным для вас образом. Как правило, я предпочитаю работать с событием `tap`. Пример обработки события, связанного с виджетом кнопки, даже если он был создан на основе элемента, не являющегося кнопкой, приведен в листинге 29.4.

Листинг 29.4. Обработка событий, связанных с виджетами кнопок

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
```

```
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $('button, #divButton').bind("tap", function() {
                $.mobile.changePage("#page2");
            })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <button>Элемент BUTTON</button>
            <div id="divButton" data-role="button">
                Элемент DIV</div>
        </div>
    </div>
    <div id="page2" data-role="page" data-theme="e">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Вы нажали кнопку!
            <a href="#" data-role="button" data-rel="back">
                OK</a>
        </div>
    </div>
</body>
</html>
```

В этом примере создается страница, содержащая элемент `button` и элемент `div`, которые оба преобразуются библиотекой jQuery Mobile в виджеты кнопок. Ответная реакция на щелчок пользователем на любой из этих кнопок организуется с помощью метода `bind()` и состоит в обработке события `tap`, описанного в главе 26. Щелчок на любой из кнопок сопровождается вызовом метода `changePage()`, приводящим к переходу на другую страницу того же документа.

Совет. Обратите внимание на то, что функции-обработчики для элементов `button` и `div` регистрируются лишь после того, как наступает событие `pageinit`. Событие `pageinit` рассматривалось в главе 26, где также было приведено краткое объяснение причин, по которым необходимо использовать именно это событие, а не событие `ready` jQuery.

Добавление значков на кнопки

Библиотека jQuery Mobile включает набор значков, предназначенных для использования на кнопках. Все значки содержатся в одном файле изображений, который находится в папке `images`, созданной вами в главе 26. Значки и их краткое описание приведены в табл. 29.3.

Таблица 29.3. Значки, включенные в библиотеку jQuery Mobile

Имя значка	*Описание*
`arrow-l`, `arrow-l`, `arrow-l`, `arrow-l`	Изображения стрелок, направленных влево, вправо, вверх и вниз
`check`, `delete`	"Галочка" и диагональный крестик
`plus`, `minus`	Знаки "плюс" и "минус"
`gear`	Шестеренка
`refresh`, `forward`, `back`, `home`, `search`	Стилизованные значки браузера, представляющие обновление страницы, переход к следующей странице, переход к предыдущей странице, возврат на домашнюю страницу и поиск
`grid`	Сетка из небольших квадратов
`star`	Звезда
`alert`	Предупреждающий знак
`info`	Стилизованная буква `i`

Чтобы применить к кнопке значок, следует указать его имя в атрибуте `data-icon`. Положение значка на кнопке можно определить с помощью атрибута `data-iconpos`. По умолчанию для этого атрибута используется значение `left`, но можно указать также значения `top`, `right` и `bottom`. Если для атрибута `data-iconpos` задано значение `notext`, то отображаться будет только значок, но не текст кнопки. Пример использования обоих атрибутов приведен в листинге 29.5.

Листинг 29.5. Добавление значков на кнопки

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div class="ui-grid-b">
                <div class="ui-block-a">
                    <button
                        data-icon="home">Домой
                    </button>
                </div>
                <div class="ui-block-b">
                    <button
```

```
                        data-icon="home" data-iconpos="top">
                        Домой
                    </button>
                </div>
                <div class="ui-block-c">
                    <button
                        data-icon="home" data-iconpos="notext">
                    </button>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере создаются три кнопки, на каждой из которых отображается значок home. Для значка первой кнопки используется позиция, установленная по умолчанию, для значка второй кнопки — позиция top, а для значка последней кнопки — значение notext, которому соответствует кнопка, представленная только значком. Как выглядят перечисленные кнопки в окне браузера, показано на рис. 29.3.

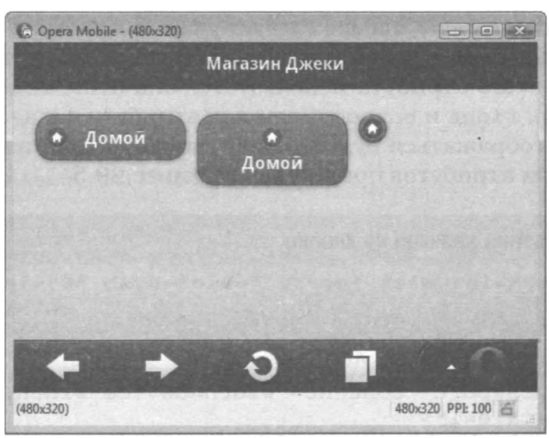


Рис. 29.3. Создание кнопок со значками

Как видите, каждая кнопка имеет характерный стиль. Обращает на себя внимание последняя кнопка, на которой текст не отображается. Визуально это выглядит привлекательно, но мой опыт работы с подобными кнопками говорит о том, что в них трудно попасть пальцем и что не все пользователи сразу же распознают в них управляющие элементы навигации, позволяющие переходить к другим частям приложения.

Создание встроенных кнопок

По умолчанию кнопки jQuery Mobile располагаются по всей ширине экрана. Пример таких кнопок был показан на рис. 29.1. В последних примерах для создания кнопок меньшего размера использовалась макетная сетка, но тот же результат можно получить с помощью *встроенных кнопок* (inline buttons), размер которых определяется размером их содержимого. Соответствующий пример приведен в листинге 29.6.

Листинг 29.6. Использование встроенных кнопок

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div>
                <button data-icon="home" data-inline=true>
                    Домой</button>
            </div>
            <div>
                <button data-icon="home">Домой</button>
            </div>
        </div>
    </div>
</body>
</html>
```

Встроенные кнопки создаются путем присвоения атрибуту `data-inline` значения `true`. В этом примере присутствуют две кнопки, одна из которых является встроенной. Результат приведен на рис. 29.4.

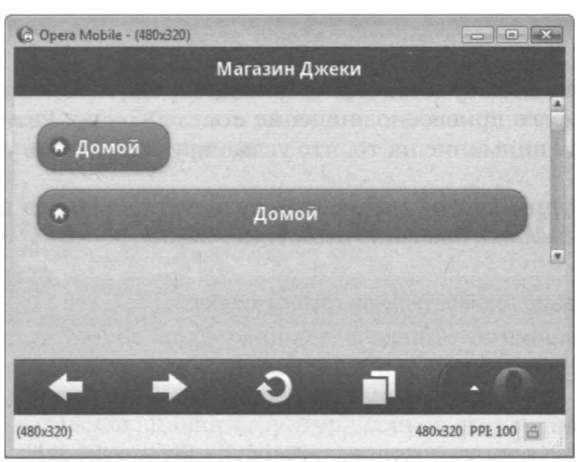


Рис. 29.4. Использование встроенных кнопок

Создание группы кнопок

Кнопки можно сгруппировать так, чтобы между ними не было промежутков, создав так называемую *управляющую группу* (control group). Это делается путем назначения атрибута `data-role` со значением `controlgroup` элементу, играющему роль родительского элемента по отношению к нескольким виджетам кнопок. Соответствующий пример приведен в листинге 29.7.

Листинг 29.7. Создание набора сгруппированных кнопок

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div data-role="controlgroup">
                <button data-icon="back">Назад</button>
                <button data-icon="home">Домой</button>
                <button data-icon="forward">Вперед</button>
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере у всех трех кнопок есть общий родитель — элемент `div`, атрибуту `data-role` которого присвоено значение `controlgroup`. Результат приведен на рис. 29.5. Обратите внимание на то, что углы скруглены лишь у верхней и нижней кнопок.

Ориентацию группы кнопок можно изменить, установив для атрибута `data-type` значение `horizontal`, как показано в листинге 29.8.

Листинг 29.8. Создание горизонтальной группы кнопок

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
```

```
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div data-role="controlgroup" data-type="horizontal"
                <button data-icon="back">Назад</button>
                <button data-icon="home">Домой</button>
                <button data-icon="forward">Вперед</button>
            </div>
        </div>
    </div>
</body>
</html>
```

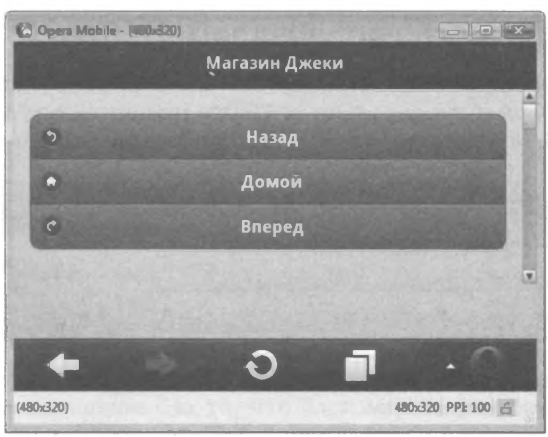


Рис. 29.5. Набор кнопок, отображаемых в виде группы

Теперь, как показано на рис. 29.6, в браузере отображается горизонтально расположенная группа кнопок. Обратите внимание, что и в данном случае скругленные углы имеют только крайние кнопки.

Использование сворачиваемых блоков содержимого

Библиотека jQuery Mobile поддерживает создание *сворачиваемых блоков содержимого* (collapsible content blocks), представляющих собой секции содержимого, снабженные заголовками. Каждую секцию можно свернуть, так что видимым остается только ее заголовок. Такая структура весьма напоминает одиночную панель виджета Accordion jQuery UI, который рассматривался в главе 19. Объединив несколько сворачиваемых блоков в единое целое, вы фактически получите виджет Accordion.

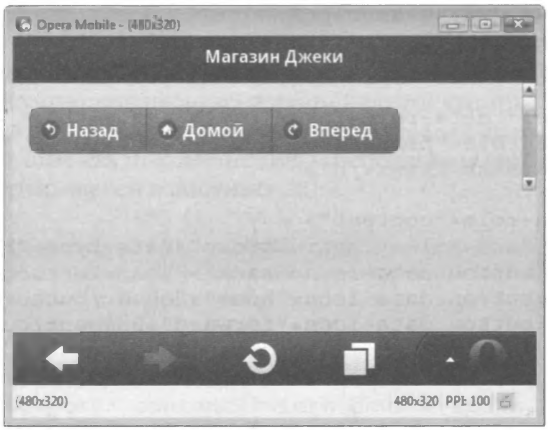


Рис. 29.6. Создание горизонтальной группы кнопок

Создание одиночного сворачиваемого блока

Сворачиваемый блок имеет специфическую структуру, которую необходимо предоставить, чтобы библиотека jQuery Mobile получила все элементы, необходимые для создания блока. Соответствующий пример приведен в листинге 29.9.

Листинг 29.9. Создание одиночного сворачиваемого блока

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div data-role="collapsible">
                <h1>Новая служба доставки</h1>
                <p>Мы рады сообщить о начале предоставления новой
                услуги - доставки заказанных вами цветов прямо на
                дом. В радиусе 20 миль от магазина доставка
                осуществляется бесплатно, доплата за каждую
                дополнительную милю составляет $1.</p>
            </div>
        </div>
    </div>
```

```
</body>
</html>
```

Прежде всего необходимо создать элемент `div` и применить к нему атрибут `data-role`, имеющий значение `collapsible`. Тем самым вы сообщаете о том, что хотите создать сворачиваемый блок, и jQuery Mobile будет искать элемент заголовка, являющийся первым дочерним элементом элемента `div`. В качестве заголовка может быть использован любой элемент от `h1` до `h6`. В примере использован элемент `h1`, но в случае данной разновидности виджетов все заголовки равноценны для jQuery Mobile. Остальные дочерние элементы элемента `div` используются в качестве содержимого сворачиваемого элемента, что дает результат, представленный на рис. 29.7.

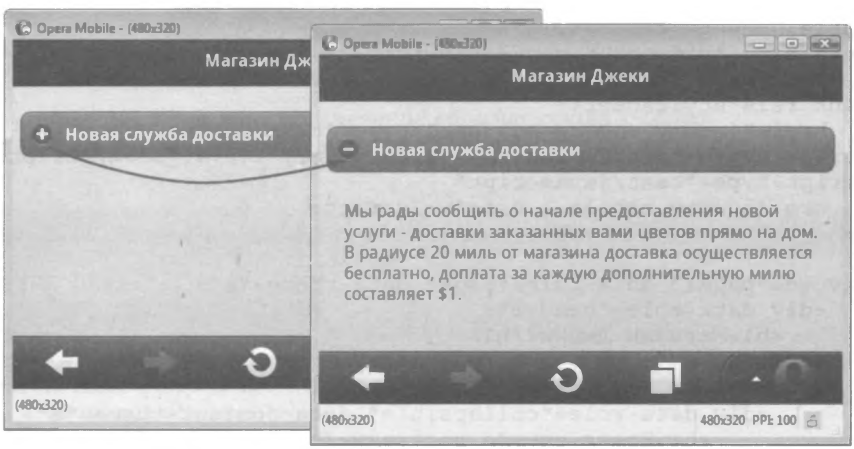


Рис. 29.7. Раскрытие сворачиваемого блока

Первоначально блок отображается в свернутом виде, т.е. его содержимое скрыто и виден лишь его заголовок. На то, что блок можно развернуть, пользователю указывает значок "+", расположенный у левого края заголовка. Щелчок на заголовке приводит к открытию содержимого и замене текущего значка значком "-", указывающим на то, что блок можно вновь свернуть.

Настройка сворачиваемых блоков содержимого jQuery Mobile

Для настройки сворачиваемых блоков в библиотеке jQuery Mobile предусмотрены два атрибута. Они описаны в табл. 29.4.

Таблица 29.4. Атрибуты данных для сворачиваемых блоков

Атрибут данных	*Описание*
`data-collapsed`	Если значение этого атрибута равно `true`, то блок отображается в свернутом виде (т.е. пользователь видит один только заголовок). Значению `false` соответствует отображение блока в развернутом виде
`data-content-theme`	Указывает тему оформления для области содержимого сворачиваемого блока

Настройка палитры для области содержимого

Можно применить к заголовку какую-либо палитру обычным способом с помощью атрибута `data-theme` и наряду с этим назначить области содержимого другую палитру, используя атрибут `data-content-theme`. Этой возможностью удобно пользоваться для создания визуального контраста между содержимым блока и его окружением, как показано в листинге 29.10.

Листинг 29.10. Назначение другой палитры для области содержимого

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div data-role="collapsible" data-content-theme="e">
                <h1>Новая служба доставки</h1>
                <p>Мы рады сообщить о начале предоставления новой
                услуги - доставки заказанных вами цветов прямо
                на дом. В радиусе 20 миль от магазина доставка
                осуществляется бесплатно, доплата за каждую
                дополнительную милю составляет $1.</p>
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере атрибут `data-content-theme` используется для назначения палитры `E` области содержимого сворачиваемого блока. Результат представлен на рис. 29.8.

Установка начального состояния

Атрибут `data-collapsed` используется для управления начальным состоянием сворачиваемого блока. Значению `false` этого атрибута соответствует отображение элементов содержимого блока. По умолчанию используется значение `true`, при котором отображается только заголовок. Пример использования этого атрибута приведен в листинге 29.11.

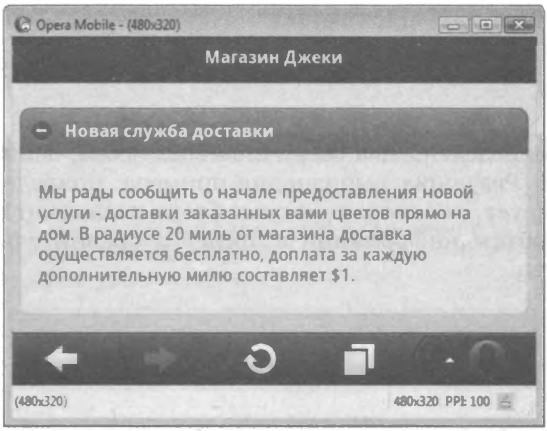


Рис. 29.8. Применение образца стиля к области содержимого сворачиваемого блока

Листинг 29.11. Установка начального состояния сворачиваемого блока

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page"
        data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div data-role="collapsible" data-collapsed=true
                data-content-theme="e">
                <h1>Новая служба доставки</h1>
                <p>Мы рады сообщить о начале предоставления новой
                услуги - доставки заказанных вами цветов прямо
                на дом. В радиусе 20 миль от магазина доставка
                осуществляется бесплатно, доплата за каждую
                дополнительную милю составляет $1.</p>
            </div>
            <div data-role="collapsible" data-collapsed=false
                data-content-theme="e">
                <h1>Специальные предложения на лето</h1>
                <p>У нас имеется множество специальных предложений
                на летний период. Подробности узнайте в магазине.
                </p>
```

```
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере определены два сворачиваемых блока, один из которых первоначально развернут. Результат выполнения примера, приведенный на рис. 29.9, наглядно демонстрирует, как сворачиваемые блоки позволяют предоставить пользователю большой объем информации в пределах сравнительно небольшого экранного пространства.

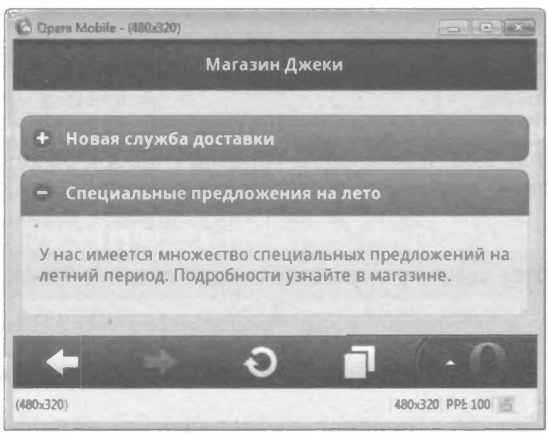


Рис. 29.9. Установка начального состояния сворачиваемого блока

Использование событий сворачиваемых блоков

Виджет сворачиваемого блока поддерживает два события: `collapse` и `expand`. Как нетрудно догадаться по их именам, эти события происходят, когда блок соответственно сворачивается или разворачивается. Пример использования этих событий приведен в листинге 29.12.

Листинг 29.12. Использование событий `collapse` и `expand`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $('#colBlock')
                .bind("collapse expand", function(event) {
                $('#status').text(event.type == "expand" ?
```

```
                    "развернут" : "свернут");
            })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            Этот блок <b><span id="status">развернут</span></b>

            <div id="colBlock" data-role="collapsible"
            data-content-theme="e" data-collapsed=false>
                <h1>Новая служба доставки</h1>
                <p>Мы рады сообщить о начале предоставления новой
                услуги - доставки заказанных вами цветов прямо на
                дом. В радиусе 20 миль от магазина доставка
                осуществляется бесплатно, доплата за каждую
                дополнительную милю составляет $1.</p>
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере для прослушивания событий `expand` и `collapse` используется метод `bind()`. Это достигается одним вызовом метода `bind()`, которому в качестве аргументов передаются сразу два события, имена которых разделены пробелом. При наступлении любого из указанных событий содержимое элемента `span` обновляется, отражая новое состояние сворачиваемого блока. Процесс изменения состояния блока проиллюстрирован на рис. 29.10.

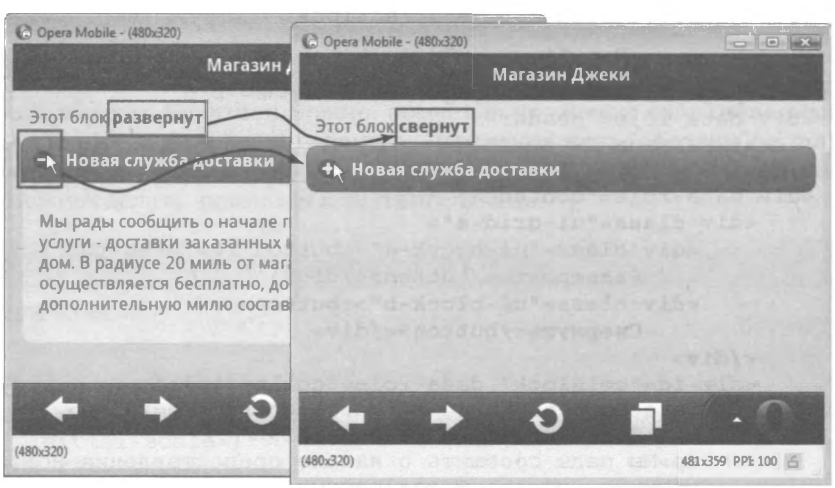


Рис. 29.10. Реагирование на события `expand` и `collapse`

Управление сворачиваемыми блоками из программы

В библиотеке jQuery Mobile не предусмотрены методы, с помощью которых сворачиваемые блоки можно было бы разворачивать и сворачивать программным способом. Однако можно самостоятельно генерировать события expand и collapse, что вызовет изменение состояния виджета. Пример самостоятельной генерации этих событий приведен в листинге 29.13.

Предупреждение. Эта возможность не документирована, и не исключено, что в будущих выпусках она предоставляться не будет. Я обнаружил ее, изучая исходный код jQuery Mobile, — занятие необычайно полезное, если вы хотите разобраться, как работает эта библиотека.

Листинг 29.13. Генерация событий для сворачивания и разворачивания блоков

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $('button').bind("tap", function() {
                var eventName = this.id == "exButton" ?
                    "expand" : "collapse";
                $('#colBlock').trigger(eventName)
            })
        });
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div class="ui-grid-a">
                <div class="ui-block-a"><button id="exButton">
                    Развернуть</button></div>
                <div class="ui-block-b"><button>
                    Свернуть</button></div>
            </div>
            <div id="colBlock" data-role="collapsible"
                data-content-theme="e" data-collapsed=false>
                <h1>Новая служба доставки</h1>
                <p>Мы рады сообщить о начале предоставления новой
                услуги - доставки заказанных вами цветов прямо на
                дом. В радиусе 20 миль от магазина доставка
                осуществляется бесплатно, доплата за каждую
```

```
                дополнительную милю составляет $1.</p>
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере я применил макетную сетку для компоновки двух кнопок и использовал метод `bind()` для прослушивания события `tap` путем запуска функции-обработчика, когда пользователь касается любой из кнопок. Для определения того, с какой именно кнопкой связано касание, используется атрибут `id`, что позволяет генерировать событие, соответствующее данной кнопке. Событие генерируется путем выбора нужного элемента средствами jQuery и последующего вызова метода `trigger()`, которому передается имя события, в данном случае `expand` или `collapse`. Далее jQuery Mobile конфигурирует виджет сворачиваемого блока так, чтобы он реагировал на эти события, как показано на рис. 29.11.

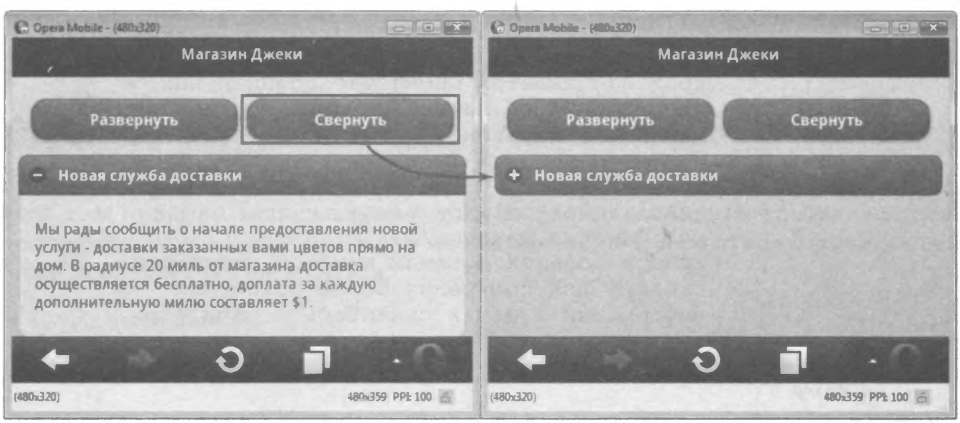


Рис. 29.11. Генерация событий для разворачивания и сворачивания блока

Создание виджетов Accordion jQuery Mobile

Создать виджет Accordion можно, поместив несколько сворачиваемых блоков в общий для них родительский элемент, играющий роль "обертки", и применив к этому элементу атрибут `data-role` со значением `collapsible-set`. Пример того, как это можно сделать, приведен в листинге 29.14.

Листинг 29.14. Создание виджета Accordion jQuery Mobile

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
```

```
        src="jquery.mobile-1.0.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">
            <div data-role="collapsible-set"
                data-content-theme="e">
                <div data-role="collapsible">
                    <h1>Новая служба доставки</h1>
                    <p>Мы рады сообщить о начале предоставления
                    новой услуги - доставки заказанных вами цветов
                    прямо на дом. В радиусе 20 миль от магазина
                    доставка осуществляется бесплатно, доплата за
                    каждую дополнительную милю составляет $1.</p>
                </div>
                <div data-role="collapsible" data-collapsed=false>
                    <h1>Специальные предложения на лето</h1>
                    <p>У нас имеется множество специальных
                    предложений на летний период. Подробности
                    узнайте в магазине.
                </p>
                </div>
                <div data-role="collapsible">
                    <h1>Оптовые заказы</h1>
                    <p>При больших объемах заказов действуют
                    скидки. Для получения более подробной
                    информации о ценах свяжитесь с нами.</p>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере определен элемент `div` с атрибутом `data-role`, имеющим значение `collapsible-set`, который содержит три сворачиваемых блока.

Совет. Обратите внимание на применение атрибута `data-content-theme` к внешнему контейнеру, что равносильно использованию данного атрибута в каждом сворачиваемом блоке по отдельности.

По умолчанию все сворачиваемые блоки находятся в свернутом состоянии, поэтому я применил к одному из них атрибут `data-collapsed` со значением `false`, чтобы при первом отображении страницы он был развернут. После щелчка на заголовке свернутого блока этот блок разворачивается, а текущий развернутый блок сворачивается. Результат проиллюстрирован на рис. 29.12.

Рис. 29.12. Разворачивание блока в виджете Accordion jQuery Mobile

Резюме

В этой главе были описаны два виджета jQuery Mobile: кнопки и сворачиваемые блоки. Как правило, использование этих виджетов вообще не требует написания пользовательского JavaScript-кода, особенно когда в качестве основы виджетов кнопок используются элементы a.

ГЛАВА 30

Использование форм jQuery Mobile

Небольшие размеры экрана крайне затрудняют визуализацию форм в мобильных устройствах. Доступная область экрана уже сама по себе мала, но при этом нужно еще предоставить пользователю элементы формы, которыми можно было бы легко манипулировать с помощью касаний, не прибегая к постоянной прокрутке экрана. В этой главе показано, как jQuery Mobile улучшает элементы формы, согласуя их вид с видом других виджетов и облегчая их использование на сенсорных экранах.

При загрузке страницы библиотека jQuery Mobile автоматически выполняет большой объем работы. Многие элементы формы улучшаются без вашего вмешательства, а при отправке формы автоматически используется Ajax, вследствие чего jQuery Mobile может без задержек получать необходимые данные с сервера.

Наряду с этим предусмотрен ряд полезных конфигурационных опций. В частности, особого внимания заслуживает элемент `select`. Можно использовать один из нескольких возможных виджетов для представления данного элемента пользователю, выбрав его по своему усмотрению, а также аккуратно обойти некоторые проблемы, связанные с собственными реализациями элементов `select` в браузерах.

Библиотека jQuery Mobile обеспечивает довольно неплохую поддержку форм, хотя эту поддержку и нельзя назвать идеальной. В частности, существуют незначительные проблемы компоновки, а именно — при компоновке страницы соседние элементы могут иметь разную ширину. Подобные проблемы иногда вызывают недовольство пользователей, однако это ничуть не уменьшает той пользы, которую приносит использование библиотеки jQuery Mobile, а если учесть наличие всеобщего интереса к данной библиотеке, то можно надеяться, что в одном из ее ближайших последующих выпусков указанные проблемы будут устранены.

Если говорить в целом, то при создании форм для экранов мобильных устройств крайне важно тщательно продумывать все детали. В силу своей природы формы предназначены для сбора данных, предоставляемых пользователями, но на небольших экранах этот процесс может быть утомительным, особенно если речь идет о вводе данных вручную. Кроме того, на большинстве мобильных устройств полосы прокрутки могут не отображаться, если пользователь прибегает к ним лишь изредка. Таким образом, пользователь может даже не догадываться о том, что за пределами отображаемой области формы могут располагаться дополнительные элементы. Чтобы сделать условия работы максимально комфортными для пользователей, необходимо придерживаться следующих основных рекомендаций.

- *Минимизируйте объем данных, вводимых с помощью клавиатуры.* Везде, где только возможно, используйте альтернативные виджеты, с помощью которых пользователь мог бы сделать необходимый выбор посредством касаний, например касаясь кнопки-флажка или кнопки-переключателя. Это сузит

диапазон величин, доступных пользователю для ввода, но может увеличить число пользователей, которые захотят заполнить форму.

- *Используйте для отображения разделов формы переходы между страницами.* Так пользователю будет легче следить за тем, какую часть формы он уже успел заполнить, и ему не потребуется прокрутка формы, дабы убедиться в том, что он ничего не пропустил.
- *Без малейших сомнений избавляйтесь от лишних элементов формы.* Формы, используемые в мобильных приложениях, следует максимально упрощать, а это означает, что от мобильных пользователей придется получать меньше данных, чем от пользователей настольных компьютеров.

Перечень тем, рассматриваемых в данной главе, приведен в табл. 30.1.

Таблица 30.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Отображение подписей рядом с элементами формы	Используйте элемент `div` с атрибутом `data-role`, равным `fieldcontain`	1
Изменение точки отсечения для стилей `fieldcontain`	Используйте медиазапрос CSS для изменения диапазона значений ширины экрана, к которому применяются стили	2
Сокрытие элементов `label`	Используйте классы `ui-hidden-accessible` и `ui-hide-label`	3, 4
Улучшение элемента `select`	Определите на странице этот элемент, и jQuery Mobile автоматически применит к нему базовое улучшение	5
Применение пользовательских списков для элемента `select`	Установите для атрибута `data-native-menu` значение `false`	6
Определение элемента-заместителя для элемента `select`	Примените к элементу `option` атрибут `data-placeholder` со значением `true`	7
Программное управление списком `select`	Используйте методы `open`, `close` и `refresh`	8
Создание ползункового переключателя	Примените к элементу `select` атрибут `data-role` со значением `slider`	9
Создание базового флажка	Определите элемент `label` и элемент `input` с типом `checkbox`	10
Применение подписей к флажкам или группе флажков	Используйте значения `fieldcontain` и `controlgroup` вместе с элементом `legend` для определения текста надписи	11, 12
Создание переключателей (радиокнопок)	Используйте ту же структуру элементов, что и для флажков	13
Создание диапазонного ползунка	Определите элемент `input` с типом `range`	14

Автоматическое создание элементов формы

В процессе обработки страницы jQuery Mobile автоматически создает виджеты для элементов формы в полной аналогии с автоматическим созданием кнопок (см. главу 29). В листинге 30.1 показана страница jQuery Mobile, которая содержит элемент `form` и некоторые его дочерние элементы, связанные с формой.

Листинг 30.1. Простая форма на странице jQuery Mobile

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Имя: </label>
                <input id="name">
            </div>
            <div data-role="fieldcontain">
                <label for="address">Адрес: </label>
                <textarea id="address"></textarea>
            </div>
            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

Это очень простая форма, но она позволяет мне подготовить сцену для демонстрации общей картины того, что происходит, когда jQuery Mobile обрабатывает формы. Здесь есть два элемента формы, `input` и `textarea`, с каждым из которых связан элемент `label`. Результат выполнения примера представлен на рис. 30.1.

Совет. Если внутри элемента `form` содержится элемент `input` с типом `submit`, то jQuery Mobile будет автоматически отправлять форму. По умолчанию это делается посредством Ajax, но такое поведение можно изменить, применив к элементу `form` атрибут `data-ajax` со значением `false`.

Работа с подписями к элементам формы

В последнем примере каждый из элементов формы вместе со своим ярлыком, или подписью, "обертывался" `div`-элементом. Для атрибута `data-role` элемента `div` я установил значение `fieldcontain`, тем самым сообщая jQuery Mobile, что элемент формы и его подпись должны выстраиваться в линию, как показано на рис. 30.1.

Рис. 30.1. Простая форма, отображаемая библиотекой jQuery Mobile

Стили, которые jQuery Mobile применяет для линейного выстраивания подписи и элемента формы, используются лишь в том случае, если ширина экрана составляет не менее 450 пикселей. При меньшем значении подпись и элемент отображаются в разных строках, как показано на рис. 30.2.

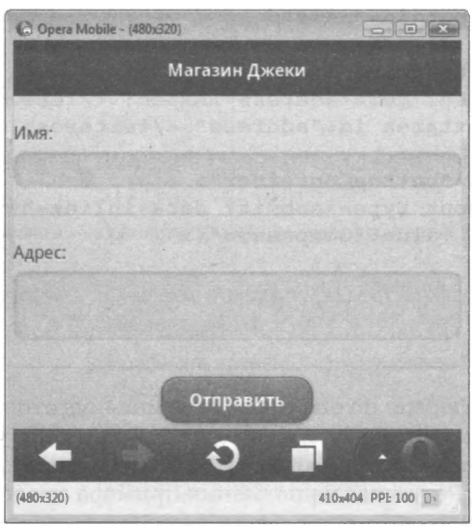


Рис. 30.2. Отображение формы в портретной ориентации страницы

Изменение точки отсечения `fieldcontain`

Установка *точки отсечения* (cutoff point) на уровне 450 пикселей меня не устраивает. Она выбрана произвольно и не учитывает широкий спектр значений ширины и разрешения экрана мобильных устройств.

Можно легко расширить область действия средства `fieldcontain`, применяя базовые CSS-классы непосредственно к документу, и, если захотите, использовать

медиазапросы для определения собственных правил относительно того, когда эти классы должны применяться. Пример использования соответствующих стилей и установки точки отсечения на уровне 100 пикселей приведен в листинге 30.2.

Листинг 30.2. Изменение точки отсечения для классов `fieldcontain`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript"
        src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
            #buttonContainer {text-align: center}

            @media all and (min-width: 100px){
                .ui-field-contain label
                .ui-input-text { vertical-align: top;
                    display: inline-block;  width: 20%;
                    margin: 0 2% 0 }
                .ui-field-contain input.ui-input-text,
                .ui-field-contain textarea.ui-input-text,
                .ui-field-contain
                .ui-input-search { width: 60%;
                     display: inline-block;}
                .ui-field-contain .ui-input-search { width: 50%; }
                .ui-hide-label input.ui-input-text,
                .ui-hide-label textarea.ui-input-text,
                .ui-hide-label .ui-input-search { padding: .4em;
                     width: 97%; }
                .ui-input-search input
                .ui-input-text { width: 98%;}
            }
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Имя: </label>
                <input id="name">
            </div>
            <div data-role="fieldcontain">
                <label for="address">Адрес: </label>
                <textarea id="address"></textarea>
            </div>
            <div id="buttonContainer">
```

```
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

Рис. 30.3. Отображение подписей рядом с элементами на узком экране

Встречая в документе атрибут `data-role` со значением `fieldcontain`, jQuery Mobile автоматически применяет нужные классы к элементам, что упрощает изменение точки отсечения или подгонку CSS-разметки.

Я уже высказывал свою точку зрения относительно произвольности установки точки отсечения на уровне 450 пикселей, но это не мешает мне согласиться с тем, что существует предельное значение ширины экрана, при котором отображение подписей рядом с элементами формы начинает мешать активизации элементов путем касания пальцами и делает невозможным просмотр содержимого. В примере я установил этот предел равным 100 пикселям, однако вы должны тщательно тестировать свои целевые устройства и самостоятельно устанавливать этот предел на таком уровне, чтобы не ухудшить удобство использования приложения. Результат выполнения примера, соответствующий уменьшению предельной допустимой ширины экрана до значения, меньшего, чем ширина окна моего эмулятора, равная 320 пикселям, приведен на рис. 30.3.

Сокрытие подписей

Один из распространенных подходов состоит в том, чтобы скрывать ярлыки, когда устройство работает в режиме отображения в портретном формате, и показывать их, когда используется альбомный формат. Я не считаю эту идею особенно привлекательной, поскольку подписи часто предоставляют важную контекстную информацию для пользователя. Но даже с учетом этой оговорки у вас всегда есть возможность предоставить дополнительные подсказки с помощью атрибута `placeholder`, появившегося в HTML5.

В библиотеке jQuery Mobile предусмотрен класс CSS, который скрывает подписи таким образом, что они остаются доступными для экранных дикторов и других вспомогательных технологий. Это достигается за счет того, что подпись остается в документе, но смещается со своего места. Чтобы скрыть отдельную подпись, к соответствующему элементу `label` применяется класс `ui-hidden-accessible`. Чтобы скрыть все подписи, находящиеся внутри элемента-контейнера, можно применить класс `ui-hide-label` к родительскому элементу. В листинге 30.3 показано, как использование обоих атрибутов совместно с атрибутами `placeholder` позволяет пользователю получить некоторое представление о том, для чего предназначен тот или иной элемент формы.

Листинг 30.3. Сокрытие подписей к элементам формы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
            #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">

            <label for="name" class="ui-hidden-accessible">
                Имя: </label>
            <input id="name" placeholder="Ваше имя">

            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="address">Адрес: </label>
                <textarea id="address"
                    placeholder="Ваш адрес"></textarea>
            </div>
            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

Совет. Обратите внимание на удаление из документа элемента с ролью `fieldcontain`, являвшегося родительским по отношению к первой паре подписи и элемента формы. Если класс `ui-hidden-accessible` применяется к элементу `label`, находящемуся внутри элемента `fieldcontain`, то ярлык, не будучи видимым, продолжает занимать отведенное для него место на экране, что ограничивает размеры элемента формы и приводит к его смещению вправо.

Результат внесения описанных изменений представлен на рис. 30.4.

Сокрытие подписей является компромиссным решением, и если устройство работает в режиме отображения с использованием альбомного формата страницы, то прибегать к этой мере обычно нет смысла. Можно реагировать на наступление события `orientationchange` (см. главу 26) и избирательно добавлять и удалять класс `ui-hide-label`, как показано в листинге 30.4. (В подобных ситуациях, чтобы удержать подпись в одной строке с элементом формы в случае альбомного формата, может потребоваться использование элементов `div` с ролью `fieldcontainer`.)

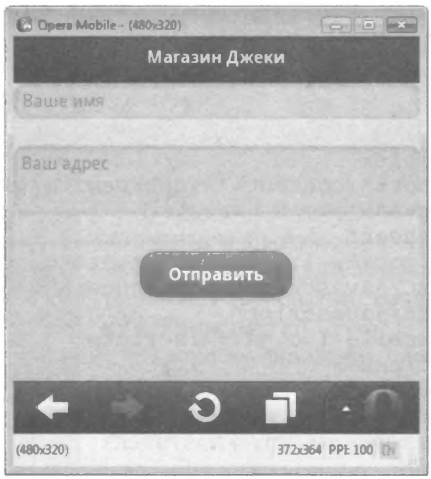


Рис. 30.4. Сокрытие подписей к элементам формы

Листинг 30.4. Избирательное сокрытие подписей

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $(window).bind("orientationchange", function(ev) {
                processOrientation(ev.orientation)
            })

            processOrientation(jQuery.event.special
                .orientationchange.orientation())

            function processOrientation(orientation) {
                var elems = $('div[data-role=fieldcontain]');
                if (orientation == "portrait") {
                    elems.addClass("ui-hide-label")
                } else {
                    elems.removeClass("ui-hide-label")
                }
            }
        })
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
```

```
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="name">Имя: </label>
                <input id="name" placeholder="Ваше имя">
            </div>
            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="address">Адрес: </label>
                <textarea id="address"
                    placeholder="Ваш адрес"></textarea>
            </div>
            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

Здесь подписи скрываются при использовании портретного формата и отображаются при использовании альбомного формата, как показано на рис. 30.5.

Рис. 30.5. Избирательное сокрытие подписей к элементам формы

Использование элементов `select`

Библиотека jQuery Mobile позволяет работать с элементами `select` двумя способами. Первый способ (представление данного элемента в виде кнопки со значком

стрелки, направленной вниз) библиотека jQuery Mobile автоматически использует, встречая в процессе обработки страницы элемент `select`. Пример использования элемента `select` приведен в листинге 30.5.

Листинг 30.5. Страница, содержащая элемент `select`

```
<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Jacqui's Shop</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Name: </label>
                <input id="name" placeholder="Your Name">
            </div>
            <div data-role="fieldcontain">
               <label for="speed"><span>Speed: </span></label>
               <select id="speed" name="speed"
                   data-native-menu=false>
                   <option value="vfast">Very Fast</option>
                   <option value="fast">Fast</option>
                   <option value="normal" selected>Normal</option>
                   <option value="slow">Slow</option>
               </select>
            </div>
            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Submit"/>
            </div>
        </form>
    </div>
</body>
</html>
```

На рис. 30.6 показано, как jQuery Mobile улучшает элементы `select`. Для этого раздела используется эмулятор Windows Phone 7, поскольку эмулятор Opera Mobile не отображает элементы `select` так, как нужно, после применения к ним стилей.

В jQuery Mobile определены некоторые атрибуты данных, которые можно использовать для точной настройки внешнего вида и поведения элементов `select`.

Эти атрибуты приведены в табл. 30.2, а смысл наиболее важных из них объясняется в следующих разделах.

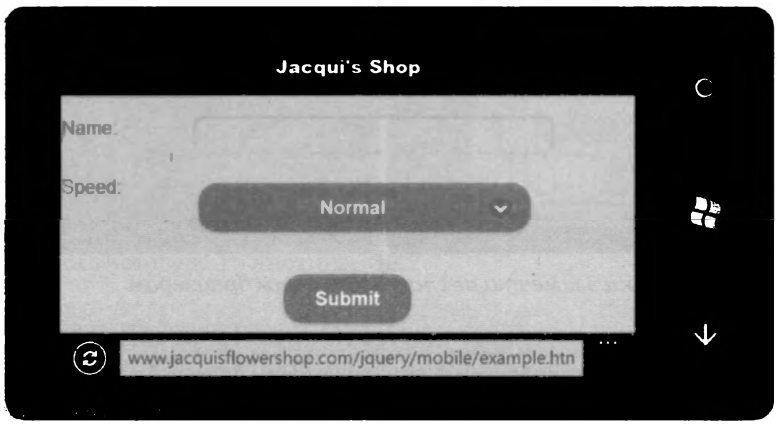


Рис. 30.6. Элемент `select`, улучшенный в jQuery Mobile

Совет. Поскольку для представления элементов `select` библиотека jQuery Mobile использует виджеты кнопок, то можно группировать элементы с помощью методики, описанной в главе 29, используя для атрибута `data-role` значение `controlgroup`.

Таблица 30.2. Атрибуты данных для элементов `select`

Атрибут	*Описание*
`data-icon`	См. описание соответствующего атрибута для виджета кнопки в главе 29
`data-iconpos`	См. описание соответствующего атрибута для виджета кнопки в главе 29
`data-inline`	См. описание соответствующего атрибута для виджета кнопки в главе 29
`data-native-menu`	Если значение этого атрибута равно `true`, то jQuery Mobile использует список `select` в том виде, в каком он реализован собственными средствами браузера. Значению `false` соответствует пользовательский список. Значение по умолчанию — `true`
`data-overlay-theme`	Определяет тему оформления для пользовательского списка `select`
`data-placeholder`	Явным образом идентифицирует элемент `option` в качестве заместителя

Применение пользовательских списков `select`

В большинстве мобильных браузеров обеспечивается сенсорно-ориентированный подход при отображении вариантов выбора элемента `select`. Например, браузер Windows Phone 7 переключается на новый экран, на котором списки опций выводятся таким образом, чтобы их можно было легко выбирать касанием пальца (рис. 30.7), тогда как в других мобильных браузерах отображается простой раскрывающийся список, размеры элементов которого достаточны для их выбора путем касания.

Можно изменить присущее браузеру поведение и поручить jQuery Mobile создание раскрывающегося списка, позволяющего выбирать опции путем касаний. Это

достигается применением атрибута `data-native-menu` со значением `false` к элементу `select`, как показано в листинге 30.6.

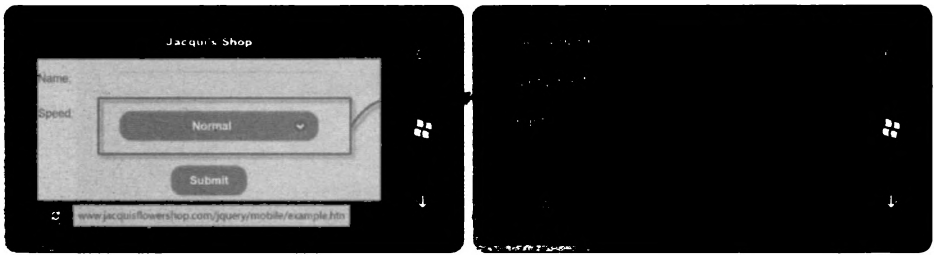


Рис. 30.7. Обработка элемента `select` мобильным браузером

Листинг 30.6. Отключение собственной функциональности браузеров, реализующей списки выбора

```
...
<div data-role="fieldcontain">
    <label for="speed"><span>Скорость: </span></label>
    <select id="speed" name="speed" data-native-menu=false>
        <option value="vfast">Очень высокая</option>
        <option value="fast">Высокая</option>
        <option value="normal" selected>Нормальная</option>
        <option value="slow">Низкая</option>
    </select>
</div>
...
```

Эта возможность полезна для применения в тех мобильных браузерах (Opera Mobile), которые не в состоянии правильно обрабатывать простые элементы `select`, а также в случаях, когда нужно создать приложение, которое выглядит одинаково в настольных и мобильных браузерах. Если для атрибута `data-native-menu` установлено значение `false`, то jQuery Mobile удаляет элемент `select` из документа и заменяет его виджетом кнопки, щелчок на которой приводит к отображению пользовательского списка, как показано на рис. 30.8.

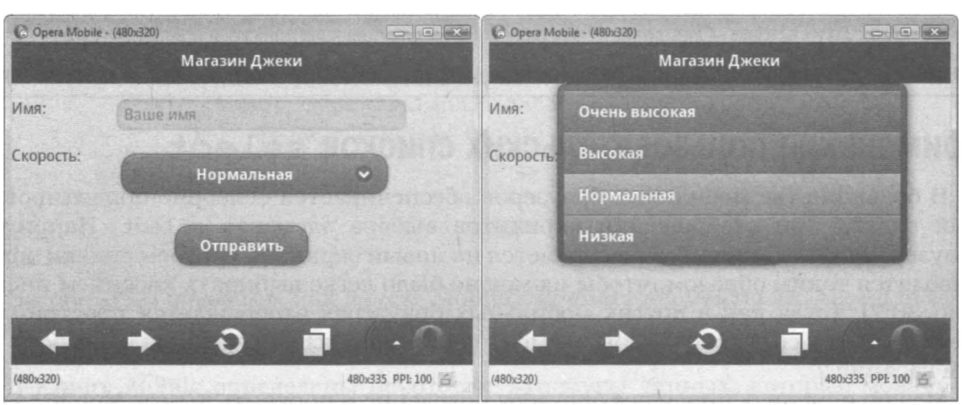


Рис. 30.8. Применение пользовательских списков jQuery Mobile в элементе `select`

Совет. По умолчанию jQuery Mobile применяет к пользовательским спискам `select` палитру A. Можно изменить это поведение, указав другую палитру в качестве нового значения атрибута `data-overlay-theme`.

Определение элементов-заместителей

Элемент `option` можно использовать в качестве заместителя элемента `select`, указав для атрибута `data-placeholder` значение `true`. Заместители появляются при первоначальном отображении элемента `select`, но не присутствуют в списке опций, предлагаемых пользователю для выбора. Пример использования атрибута `data-placeholder` приведен в листинге 30.7.

Листинг 30.7. Использование атрибута `data-placeholder`

```
...
<div data-role="fieldcontain">
    <label for="speed"><span>Скорость: </span></label>
    <select id="speed" name="speed" data-native-menu=false>
        <option data-placeholder=true value="placeholder">
            Выбор скорости</option>
        <option value="vfast">Очень высокая</option>
        <option value="fast">Высокая</option>
        <option value="normal">Нормальная</option>
        <option value="slow">Низкая</option>
    </select>
</div>
```

Результат выполнения примера показан на рис. 30.9. Вообще-то мне нравится использовать заместители совместно со списками `select`, но в данном случае эта методика особенно полезна для предоставления пользователю информации о контексте, когда элементы `label` скрываются в случае портретного формата отображения.

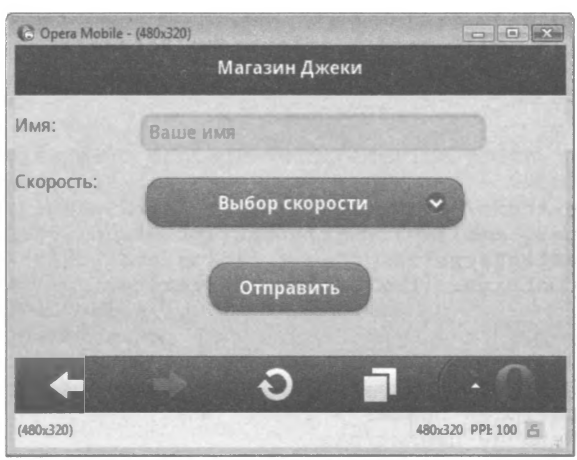


Рис. 30.9. Определение элемента-заместителя

Программное управление списком `select`

Списком выбора можно управлять из программы с помощью методов, описанных в табл. 30.3. Эти методы следуют принятому в jQuery UI стандарту передачи строки методу виджета, каковым в данном случае является метод `selectmenu`.

Таблица 30.3. Методы для элементов `select`

Метод	*Описание*
`selectmenu("open")`	Открывает список `select`
`selectmenu("close")`	Закрывает список `select`
`selectmenu("refresh")`	Обновляет виджет для включения изменений в базовый элемент `select`

Пример использования кнопок для управления списком `select` приведен в листинге 30.8.

Листинг 30.8. Управление списком `select` из программы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript">
        $(document).bind("pageinit", function() {
            $('button').bind("tap", function() {
                if (this.id == "open") {
                    $('#speed').selectmenu("open");
                    return false;
                } else {
                    $('#speed').selectmenu("close");
                    return false;
                }
            })
        })
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
```

```
            <div class="ui-grid-a">
                <div class="ui-block-a">
                    <button id="open">Открыть меню</button>
                </div>
                <div class="ui-block-b">
                    <button id="close">Закрыть меню</button>
                </div>
            </div>

            <select id="speed" name="speed"
                data-native-menu=false>
                <option data-placeholder=true
                    value="placeholder">Выбор скорости</option>
                <option value="vfast">Очень высокая</option>
                <option value="fast">Высокая</option>
                <option value="normal">Нормальная</option>
                <option value="slow">Низкая</option>
            </select>

            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

Создание ползунковых переключателей

Если элемент `select` содержит только два элемента `option`, то вместо обычного списка выбора можно создать *ползунковый переключатель* (flip switch). Для этого следует применить к элементу `select` атрибут `data-role` со значением `slider`, как показано в листинге 30.9.

Листинг 30.9. Создание ползункового переключателя

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
```

```
        </div>
        <form method="get">

            <div data-role="fieldcontain">
                <label for="speed"><span>Скорость: </span></label>
                <select id="speed" name="speed"
                    data-role="slider">
                    <option value="fast">Высокая</option>
                    <option value="slow">Низкая</option>
                </select>
            </div>

            <div data-role="fieldcontain">
                <label for="size"><span>Размер: </span></label>
                <select id="size" name="size" data-role="slider">
                    <option value="large">Крупный</option>
                    <option value="small" selected>Мелкий</option>
                </select>
            </div>

            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

В этом примере присутствуют два переключателя. На рис. 30.10 показано, как они отображаются в браузере. Пользователь может изменить настройку постукиванием пальца или щелчком мыши на открытом значении параметра либо путем перетаскивания ползунка переключателя в нужное положение.

Рис. 30.10. Использование ползунковых переключателей

Создание флажков

Настроить внешний вид флажков можно несколькими способами. Простейший из них — предоставить jQuery Mobile элемент `input`, имеющий тип `checkbox`, за которым следует элемент `label`, как показано в листинге 30.10.

Листинг 30.10. Создание простых флажков

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet" href="jquery.mobile-1.0.css"
        type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Имя: </label>
                <input id="name" placeholder="Ваше имя">
            </div>

            <input type="checkbox" name="check" id="check"/>
            <label for="check">Я согласен</label>

            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

Как выглядит такой флажок, показано на рис. 30.11. На рисунке представлены два состояния флажка — при установленной и снятой отметке.

Применение подписи к флажку

По умолчанию флажки располагаются по всей ширине родительского элемента, в нашем примере — по всей ширине экрана. Если вы хотите, чтобы флажок был выровнен в макете по располагающемуся над ним текстовому полю, используйте специальную структуру элементов, как показано в листинге 30.11.

Рис. 30.11. Простой флажок jQuery Mobile

Листинг 30.11. Изменение компоновки флажка

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Имя: </label>
                <input id="name" placeholder="Ваше имя">
            </div>

            <div data-role="fieldcontain">
                <fieldset data-role="controlgroup">
                    <legend>Сроки и условия:</legend>
                    <input type="checkbox" name="check"
                        id="check"/>
                    <label for="check">Я согласен</label>
                </fieldset>
            </div>

            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
```

```
            </div>
        </form>
    </div>
</body>
</html>
```

С наружным элементом вы уже хорошо знакомы — это элемент `div`, атрибуту `data-role` которого присвоено значение `fieldcontain`. Проблема, с которой здесь сталкивается jQuery Mobile, состоит в том, что уже существует элемент `label`, связанный с элементом `input`, и поэтому нужно найти альтернативный способ предоставления jQuery Mobile необходимой информации. Это достигается за счет добавления элемента `fieldset`, атрибуту `data-role` которого присваивается значение `controlgroup`, а также элемента `legend`, содержащего необходимый для отображения текст, перед элементом `input`. Результат представлен на рис. 30.12. Новый вариант компоновки флажка не совсем идеален, но он значительно лучше прежнего.

Рис. 30.12. Изменение компоновки флажка на странице

Группировка флажков

Элемент `fieldset` с атрибутом `data-role`, имеющим значение `contolgroup`, можно также использовать для объединения нескольких флажков в группу. Соответствующий пример приведен в листинге. 30.12.

Листинг 30.12. Группировка флажков

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
```

```
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Имя: </label>
                <input id="name" placeholder="Ваше имя">
            </div>

            <div data-role="fieldcontain">
                <fieldset data-role="controlgroup">
                    <legend>Выберите цветы:</legend>
                    <input type="checkbox"
                        name="roses" id="roses"/>
                    <label for="roses">Розы</label>
                    <input type="checkbox"
                        name="orchids" id="orchids"/>
                    <label for="orchids">Орхидеи</label>
                    <input type="checkbox"
                        name="astors" id="astors"/>
                    <label for="astors">Астры</label>
                </fieldset>
            </div>

            <div data-role="fieldcontain">
                <fieldset data-role="controlgroup"
                    data-type="horizontal">
                    <legend>Шрифт:</legend>
                    <input type="checkbox" name="bold" id="bold"/>
                    <label for="bold"><b>b</b></label>
                    <input type="checkbox"
                        name="italic" id="italic"/>
                    <label for="italic"><em>i</em></label>
                    <input type="checkbox" name="underline"
                        id="underline"/>
                    <label for="underline"><u>u</u></label>
                </fieldset>
            </div>

            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

В этом примере присутствуют две группы переключателей. Первый набор переключателей имеет вертикальную конфигурацию, которая используется по умолчанию. Стиль виджетов автоматически изменяется так, чтобы отдельные элементы `input` располагались без промежутков между ними, а скругленными были лишь внешние углы блока. Для второго набора определен атрибут `data-type`, имеющий значение `horizontal`, что приводит к изменению направления раскладки флажков и вынуждает jQuery Mobile скрыть "галочку", в результате чего создается набор кнопок, которые могут находиться во включенном или выключенном состоянии. Результат представлен на рис. 30.13.

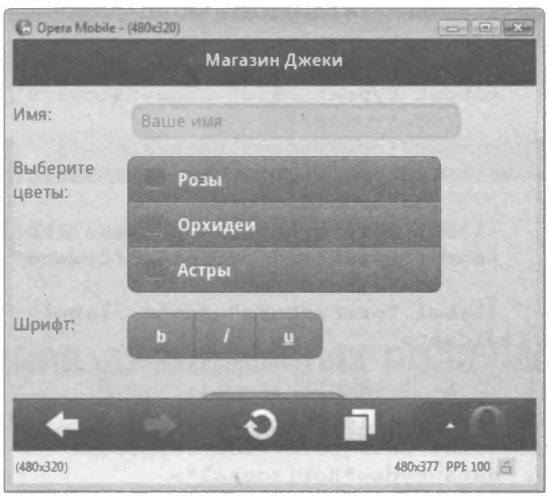


Рис. 30.13. Группировка флажков

Создание переключателей

Переключатели (радиокнопки) форматируются в основном так же, как и флажки. Соответствующий пример приведен в листинге 30.13.

Листинг 30.13. Создание группы переключателей

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
```

```
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Имя: </label>
                <input id="name" placeholder="Ваше имя">
            </div>

            <div data-role="fieldcontain">
                <fieldset data-role="controlgroup">
                    <legend>Выберите цветы:</legend>
                    <input type="radio" name="flowers"
                        id="roses"/>
                    <label for="roses">Розы</label>
                    <input type="radio" name="flowers"
                        id="orchids"/>
                    <label for="orchids">Орхидеи</label>
                    <input type="radio" name="flowers"
                        id="astors"/>
                    <label for="astors">Астры</label>
                </fieldset>
            </div>

            <div data-role="fieldcontain">
                <fieldset data-role="controlgroup"
                    data-type="horizontal">
                    <legend>Выберите цветы:</legend>
                    <input type="radio" name="flowers"
                        id="roses"/>
                    <label for="roses">Розы</label>
                    <input type="radio" name="flowers"
                        id="orchids"/>
                    <label for="orchids">Орхидеи</label>
                    <input type="radio" name="flowers"
                        id="astors"/>
                    <label for="astors">Астры</label>
                </fieldset>
            </div>

            <div id="buttonContainer">
                <input type="submit" data-inline="true"
                    value="Отправить"/>
            </div>
        </form>
    </div>
</body>
</html>
```

Здесь, как и в предыдущем примере, созданы горизонтальная и вертикальная группы переключателей, вид которых в окне браузера представлен на рис. 30.14.

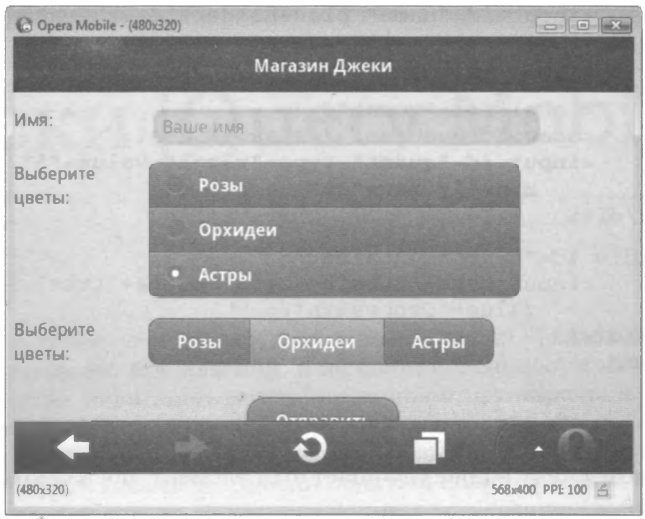


Рис. 30.14. Создание группы переключателей

Использование диапазонных ползунков

Встретив элемент `input` с типом `range`, jQuery Mobile создает *диапазонный ползунок* (range slider). Пример использования этого элемента на странице приведен в листинге 30.14.

Листинг 30.14. Использование диапазонного ползунка

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        #buttonContainer {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <form method="get">

            <div data-role="fieldcontain">
                <label for="name">Имя: </label>
```

```
            <input id="name" placeholder="Ваше имя">
        </div>

        <div data-role="fieldcontain">
            <label for="quant">
            <span>Количество: </span></label>
            <input id="quant" type="range" value="5"
                min="1" max="10" />
        </div>

        <div id="buttonContainer">
            <input type="submit" data-inline="true"
                value="Отправить"/>
        </div>
    </form>
</div>
</body>
</html>
```

Каким образом jQuery Mobile улучшает этот элемент, показано на рис. 30.15.

Рис. 30.15. Использование диапазонного ползунка

Резюме

В этой главе было показано, как jQuery Mobile улучшает элементы формы, придавая им внешний вид, гораздо лучше приспособленный к сенсорно-ориентированному стилю представления. Для отправки формы не нужно предпринимать никаких действий — это делается автоматически с помощью Ajax, что позволяет jQuery Mobile без задержек переходить на страницу, возвращаемую сервером. Можно полностью полагаться на jQuery Mobile во всем, что касается автоматического улучшения элементов формы, но могут находиться и веские причины для того, чтобы применять некоторые дополнительные элементы и атрибуты `data-role`, особенно если речь идет о работе с элементами `select`. В целом, jQuery Mobile обрабатывает элементы элегантно и изящно, хотя иногда и могут проявляться некоторые незначительные проблемы, связанные с недостаточно корректной обработкой размеров элементов относительно размера экрана.

ГЛАВА 31

Списки jQuery Mobile

В этой главе речь пойдет о списках jQuery Mobile. Списки — важный инструмент построения мобильных веб-приложений, и их часто используют в качестве простого и наглядного средства навигации между различными функциональными областями веб-приложения. Преимуществом списков является то, что они компактны, даже если их отдельные элементы имеют достаточно большие размеры для того, чтобы их можно было выбирать касанием. Списки понятны пользователям. Уже одного помещения значка стрелки справа от элемента списка (поведение jQuery Mobile по умолчанию) для большинства пользователей достаточно, чтобы понять, что выделение элемента позволяет осуществить некий выбор или перейти к другой части документа. Перечень тем, рассматриваемых в данной главе, приведен в табл. 31.1.

Таблица 31.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Создание навигационного списка jQuery Mobile	Определите элемент `ul` или `ol`, который содержит один или несколько элементов `li` и имеет атрибут `data-role`, равный `listview`. Содержимым элементов `li` должны быть ссылки	1
Создание списков, предназначенных только для чтения, или стандартных списков HTML	Создайте список, предназначенный только для чтения, опустив ссылки в элементах `li`. Создайте стандартный список HTML, опустив атрибут `data-role`	2
Создание вставного списка	Установите для атрибута `data-inset` значение `true`	3
Создание списка, элементы которого состоят из двух отдельных частей	Добавьте вторую ссылку в каждый из элементов `li`	4
Предоставление пользователю возможности осуществлять фильтрацию содержимого списка	Установите для атрибута `data-filter` значение `true`	5, 6
Добавление разделителей в список	Установите для атрибутов `data-role` отдельных элементов `li` значение `list-divider`	7
Добавление значка номера в элемент списка	Используйте класс `ui-li-count`	8
Использование различных способов выделения текста	Используйте элементы `h1-h6` и `p`	9
Добавление боковой вставки в элемент списка	Используйте класс `ui-li-aside`	10

Приступаем к работе со списками

Для создания списков jQuery Mobile существуют несколько способов. В листинге 31.1 показано, как создать простой список, ссылки которого указывают на страницы jQuery Mobile, принадлежащие тому же документу. Каждая страница описывает отдельный вид цветов, и список служит тем механизмом, с помощью которого пользователь может переходить к нужным страницам.

Листинг 31.1. Простой список

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        .lcontainer {float: left; text-align: center;
             padding-top: 10px}
        .productData {float: right; padding: 10px; width: 60%}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>

        <ul data-role="listview">
            <li><a href="#roses">Розы</a></li>
            <li><a href="#orchids">Орхидеи</a></li>
            <li><a href="#astors">Астры</a></li>
        </ul>
    </div>

    <div id="roses" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Розы</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="rose.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                Роза - многолетнее древовидное растение семейства
                розоцветных, произрастающее в виде кустов с
                вертикальными стеблями и вьющимися или плетистыми
```

```
                побегами.
                <div><b>Цена: $4.99</b></div>
            </div>
        </div>
    </div>

    <div id="orchids" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Орхидеи</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="orchid.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                Орхидные - многообразное и широко распространенное
                семейство порядка спаржецветных. Это одно из самых
                крупных семейств цветущих растений.
                <div><b>Цена: $10.99</b></div>
            </div>
        </div>
    </div>

    <div id="astors" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Астры</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="astor.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                Название "астра" происходит от древнегреческого
                слова, означающего "звезда", и объясняется формой
                соцветия.
                <div><b>Цена: $2.99</b></div>
            </div>
        </div>
    </div>
</body>
</html>
```

Большая часть этого документа представлена страницами, описывающими цветы. Собственно список состоит всего лишь из нескольких элементов.

```
<ul data-role="listview">
    <li><a href="#roses">Розы</a></li>
    <li><a href="#orchids">Орхидеи</a></li>
    <li><a href="#astors">Астры</a></li>
</ul>
```

Совет. В этом примере используется список `ul`, но точно так же jQuery Mobile обрабатывает и нумерованные списки (которые создаются с помощью элемента `ol`).

Это стандартный ненумерованный список HTML, созданный на основе элемента `ul`, содержащего три элемента `li`. Для создания виджета списка jQuery Mobile атрибуту `data-role` элемента `ul` присвоено значение `listview`. Виджет списка в основном используется в качестве средства навигации, и поэтому каждый из элементов `li` содержит элемент `a`, ссылающийся на одну из других страниц документа. Вид этого списка в окне браузера представлен на рис. 31.1.

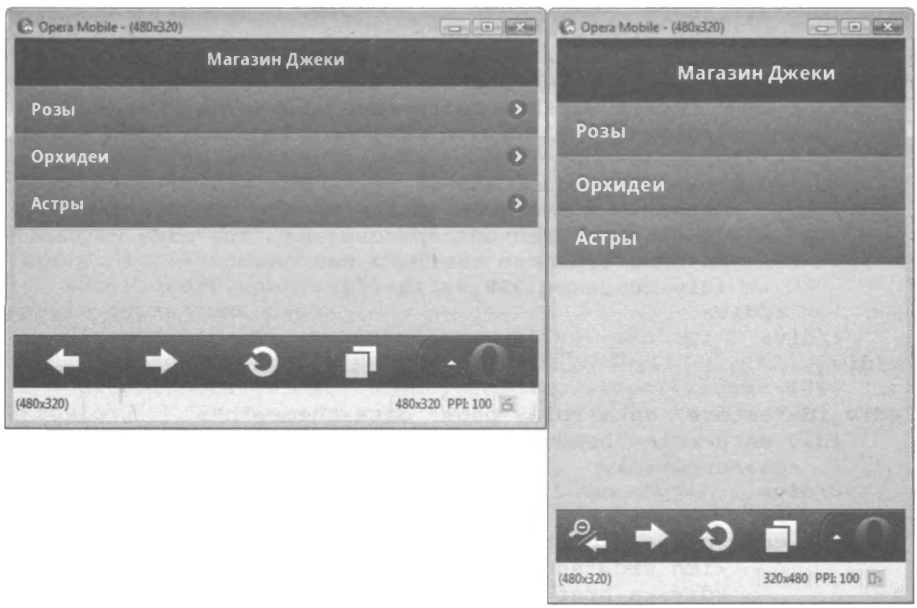


Рис. 31.1. Простой виджет списка jQuery Mobile

Щелчок или нажатие на элементе списка вызывают переход на соответствующую страницу. Одна из этих страниц показана на рис. 31.2. На каждую страницу добавлена кнопка, позволяющая вернуться к списку.

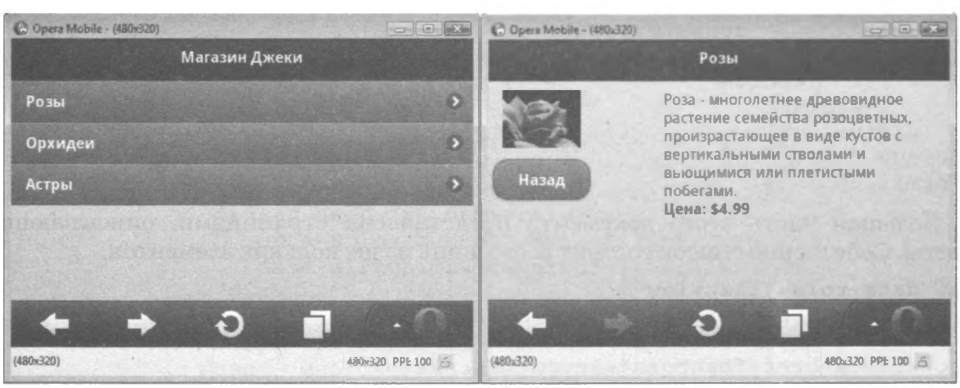


Рис. 31.2. Использование списка для перехода на другие страницы документа

Форматирование списков

В jQuery Mobile определен ряд атрибутов данных, которые можно использовать для форматирования списков. Эти атрибуты перечислены в табл. 31.2.

Таблица 31.2. Атрибуты для форматирования списков

Атрибут	*Описание*
`data-filter`	Если значение этого атрибута равно `true`, то отображается текстовое поле, с помощью которого пользователь может осуществлять фильтрацию списка. Значение по умолчанию — `false`
`data-filter-placeholder`	Определяет строку-заместитель, которая будет отображаться в текстовом поле, если значение атрибута `data-filter` равно `true`
`data-filter-theme`	Определяет палитру, которая будет применяться к текстовому полю, если значение атрибута `data-filter` равно `true`
`data-inset`	Если значение этого атрибута равно `true`, то список будет отображаться со скругленными углами. Значению по умолчанию, равному `false`, соответствуют прямые углы
`data-split-icon`	Определяет значок разделителя между кнопками

Создание элементов простого списка

В последнем примере был продемонстрирован список элементов, содержащих элементы `a`, поскольку это наиболее распространенный способ использования виджета списка. Опустив эти ссылки, можно получить простой список, называемый *списком, предназначенным только для чтения* (read-only list). Если же опустить в элементах `ul` и `ol` атрибут `data-role`, то jQuery Mobile вообще не будет создавать виджет, и вы получите стандартный список HTML. Пример использования обоих способов приведен в листинге 31.2.

Листинг 31.2. Создание неформатированных списков и списков, предназначенных только для чтения

```
<div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
       <h1>Магазин Джеки</h1>
    </div>

    <ul>
        <li><a href="#roses">Розы</a></li>
        <li><a href="#orchids">Орхидеи</a></li>
        <li><a href="#astors">Астры</a></li>
    </ul>

    <ul data-role="listview" data-theme="e">
        <li>Розы</li>
        <li>Орхидеи</li>
        <li>Астры</li>
    </ul>
```

```
    </div>
...
```

Первый список — это типичный список HTML. Я включил его в документ, чтобы даже при отсутствии виджета jQuery Mobile пользователь мог использовать ссылки для перехода на другие страницы. В этом и состоит суть базового подхода к организации навигации, с которым вы познакомились в главе 27.

Второй список предназначен только для чтения. Когда вы создаете список этого типа, jQuery Mobile применяет к содержимому используемые по умолчанию стили из текущей палитры, и поэтому список теряется на фоне остального содержимого страницы. Чтобы сделать список более заметным, я использовал атрибут `data-theme` в элементе `ul`, в результате чего элементы списка выделяются другой палитрой. Оба списка представлены на рис. 31.3.

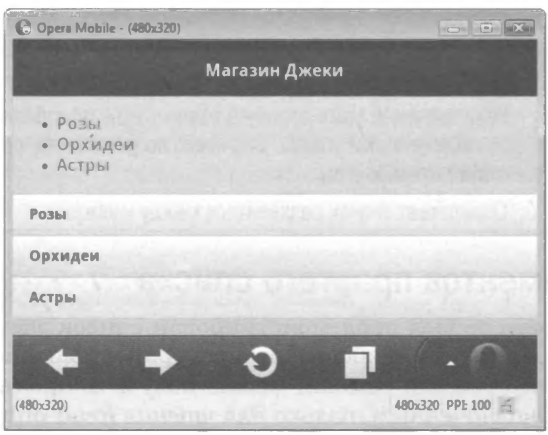


Рис. 31.3. Простой список HTML и список, предназначенный только для чтения

Я рекомендую осторожно относиться к использованию списков, предназначенных только для чтения, особенно в тех случаях, когда в приложении используются другие списки, с помощью которых осуществляется навигация по приложению. Списки, предназначенные только для чтения, не имеют достаточно заметных внешних отличительных признаков, и, как показывают результаты проводившегося мною тестирования удобства использования, пользователи пытаются применять эти списки для навигации и, не видя никакой ответной реакции, теряются и выражают недовольство.

Создание вставных списков

В используемой по умолчанию компоновке список располагается по всей ширине контейнерного элемента и имеет прямые углы, что не соответствует стилю остальных виджетов jQuery Mobile. Чтобы обеспечить единообразие стилей, можно создать *вставной список* (inset list), который имеет скругленные углы и может использоваться в элементах, не достигающих краев экрана. Для этого следует применить к элементам `ul` и `ol` атрибут `data-inset` и присвоить ему значение `true`, как показано в листинге 31.3.

Листинг 31.3. Создание вставного списка

```
...
<div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
       <h1>Магазин Джеки</h1>
    </div>

    <div id="container" style="padding: 20px">
        <ul data-role="listview" data-inset=true>
            <li><a href="#roses">Розы</a></li>
            <li><a href="#orchids">Орхидеи</a></li>
            <li><a href="#astors">Астры</a></li>
        </ul>
    </div>
</div>
...
```

В этом примере элемент `ul` находится внутри элемента `div`. Для создания промежутков между краями списка и краями родительского элемента использован параметр CSS `padding`, а для изменения стиля списка — атрибут `data-inset`. Результат представлен на рис. 31.4.

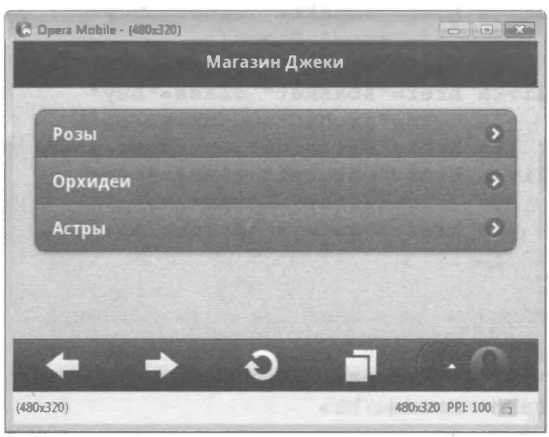


Рис. 31.4. Создание вставных списков

Создание разделенных списков

В тех случаях, когда с каждым элементом списка могут быть связаны два различных действия, можно использовать *разделенные списки* (split list). Элемент списка разбивается на две части, щелчки или касания на которых приводят к разным действиям. Пример разделенного списка, с помощью которого пользователь может получить информацию о цветке или просто добавить его в корзину, приведен в листинге 31.4.

Листинг 31.4. Использование разделенного списка

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        .lcontainer {float: left; text-align: center;
            padding-top: 10px}
        .productData {float: right; padding: 10px; width: 60%}
        .cWrapper {text-align: center; margin: 20px}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Jacqui's Shop</h1>
        </div>

        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true>
                <li><a href="#basket" class="buy"
                        id="rose">Розы</a>
                    <a href="#roses">Розы</a></li>
                <li><a href="#basket" class="buy"
                        id="orchid">Орхидеи</a>
                    <a href="#orchids">Орхидеи</a></li>
                <li><a href="#basket" class="buy"
                        id="astor">Астры</a>
                    <a href="#astors">Астры</a></li>
            </ul>
        </div>
    </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div class="cWrapper">
            Здесь будет находиться корзина покупателя
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back"
                data-role="button" data-inline=true
                data-direction="reverse">Назад</a>
        </div>
    </div>

    <div id="roses" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Розы</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="rose.png">
```

```
            <div><a href="#" data-rel="back"
                data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
            </div>
        </div>
        <div class="productData">
            Роза - многолетнее древовидное растение семейства
            розоцветных, произрастающее в виде кустов с
            вертикальными стеблями и вьющимися или плетистыми
            побегами.
            <div><b>Цена: $4.99</b></div>
        </div>
    </div>
</div>

<div id="orchids" data-role="page" data-theme="b">
    <div data-role="header">
       <h1>Орхидеи</h1>
    </div>
    <div>
        <div class="lcontainer">
            <img src="orchid.png">
            <div><a href="#" data-rel="back"
                data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
            </div>
        </div>
        <div class="productData">
            Орхидные - многообразное и широко распространенное
            семейство порядка спаржецветных. Это одно из самых
            крупных семейств цветущих растений.
            <div><b>Цена: $10.99</b></div>
        </div>
    </div>
</div>

<div id="astors" data-role="page" data-theme="b">
    <div data-role="header">
       <h1>Астры</h1>
    </div>
    <div>
        <div class="lcontainer">
            <img src="astor.png">
            <div><a href="#" data-rel="back"
                data-role="button" data-inline=true
                data-direction="reverse">Назад</a>
            </div>
        </div>
        <div class="productData">
              Название "астра" происходит от древнегреческого
              слова, означающего "звезда", и связано с формой
              соцветия.
            <div><b>Цена: $2.99</b></div>
        </div>
    </div>
</div>
</body>
</html>
```

Создание разделенного списка не составляет большого труда. Для этого нужно всего лишь добавить ко второму элементу в элементы `li`. В примере разделенный список выделен полужирным шрифтом. Между элементами списка, образующими пару, jQuery Mobile вставляет разделитель в виде вертикальной черты. Щелчок на левом или правом элементе пары вызывает переход по ссылке, указанной в соответствующем элементе `a`. Внешний вид списка в окне браузера представлен на рис. 31.5.

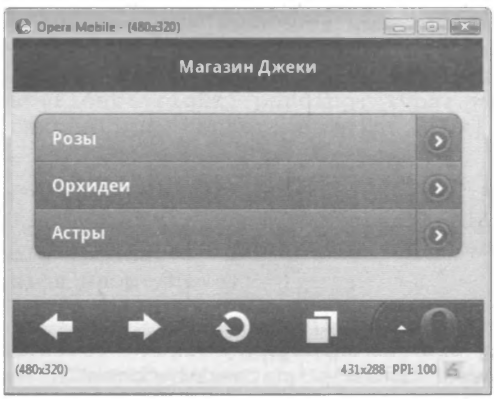


Рис. 31.5. Создание разделенных списков

В этом примере левые части элементов списка указывают на добавленную в документ новую страницу `basket`. Мы вернемся к этому примеру в главе 32 и добавим в него недостающий код корзины покупателя. В данном примере вместо страницы корзины используется ее заместитель.

Совет. По умолчанию для кнопок разделенных списков используется значок стрелки. Можно заменить этот значок, применив атрибут `data-split-icon` к элементу `ul` или `ol` и указав в нем имя другого значка.

Фильтрация списков

Библиотека jQuery Mobile предоставляет механизм для фильтрации содержимого списков. Фильтрация списков активизируется путем применения атрибута `data-filter` со значением `true` к элементу `ul` или `ol`, как показано в листинге 31.5.

Листинг 31.5. Фильтрация списков

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
```

```
        .lcontainer {float: left; text-align: center;
            padding-top: 10px}
        .productData {float: right; padding: 10px; width: 60%}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>

        <div data-role="content">
            <ul data-role="listview" data-inset=true
                data-filter=true
                data-filter-placeholder="Фильтрация списка...">
                <li><a href="#roses">Розы</a></li>
                <li><a href="#orchids">Орхидеи</a></li>
                <li><a href="#astors">Астры</a></li>
            </ul>
        </div>
    </div>

  <div id="roses" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Розы</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="rose.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                Роза - многолетнее древовидное растение семейства
                розоцветных, произрастающее в виде кустов с
                вертикальными стеблями и вьющимися или плетистыми
                побегами.
                <div><b>Цена: $4.99</b></div>
            </div>
        </div>
    </div>

   <div id="orchids" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Орхидеи</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="orchid.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                Орхидные - многообразное и широко распространенное
```

```
                семейство порядка спаржецветных. Это одно из самых
                крупных семейств цветущих растений.
                <div><b>Цена: $10.99</b></div>
            </div>
        </div>
    </div>

    <div id="astors" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Астры</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="astor.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                  Название "астра" происходит от древнегреческого
                  слова, означающего "звезда", и связано с формой
                  соцветия.
                <div><b>Цена: $2.99</b></div>
            </div>
        </div>
    </div>
</body>
</html>
```

Как показано на рис. 31.6, jQuery Mobile добавляет над списком поле поиска. В процессе ввода пользователем символов в поле поиска jQuery Mobile удаляет из списка все элементы, в которых данная последовательность символов не встречается.

Предупреждение. В случае фильтрации нумерованного списка jQuery Mobile не сохраняет номера элементов.

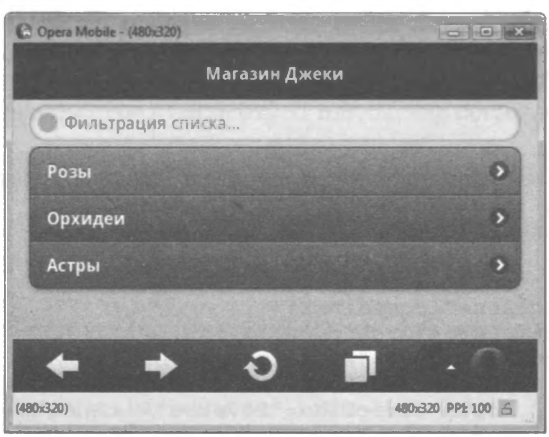


Рис. 31.6. Активизация фильтрации списка

Совет. Несмотря на то что обычно элемент `content` на страницах jQuery Mobile можно безболезненно опускать, в случае фильтрации это не так. Без этого элемента размер и положение поля поиска будут обрабатываться некорректно.

Фильтрация списков — замечательная функция, но на небольших экранах она не всегда полезна. В большинстве мобильных устройств для поддержки символьного ввода предусмотрена виртуальная сенсорная клавиатура, которая отображается всякий раз, когда пользователь активизирует элемент текстового ввода, например поле поиска. В случае небольших устройств клавиатура занимает значительную часть экрана, тем самым не давая пользователю возможности увидеть результаты фильтрации, как показано на рис. 31.7. Отсюда вовсе не следует, что вы не должны поддерживать фильтрацию списков, но в подобных случаях, когда размер экранов целевых устройств невелик, очень важно предоставлять в распоряжение пользователя другие навигационные механизмы.

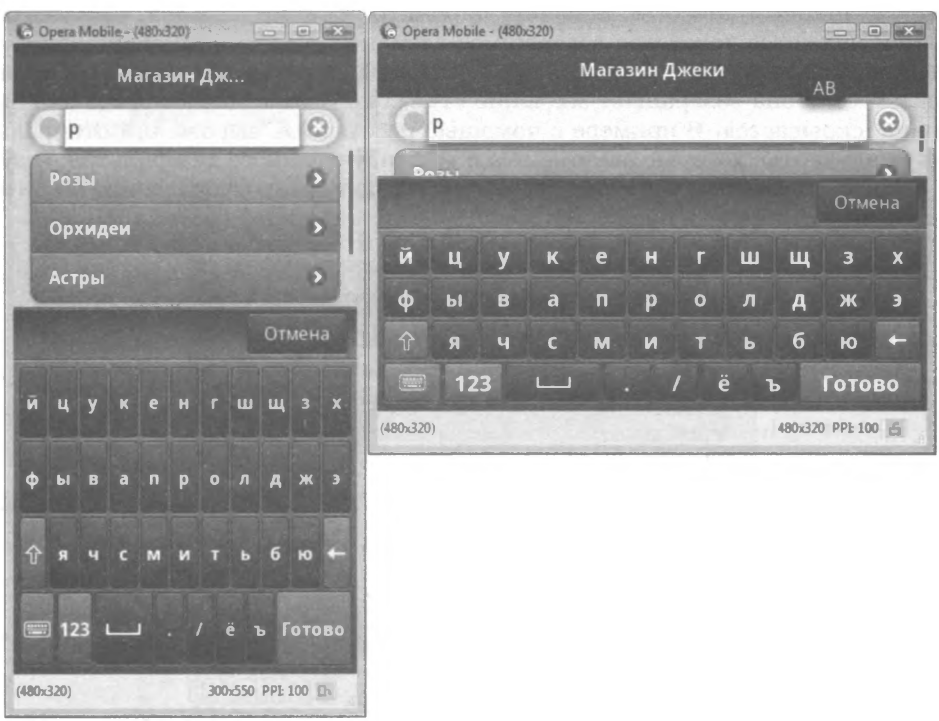


Рис. 31.7. Сенсорная клавиатура, заслоняющая отфильтрованный список

Пользовательские функции фильтрации

По умолчанию фильтру соответствуют элементы списка, которые содержат набор символов, введенных пользователем. Соответствия ищутся в любом месте элемента списка без учета регистра набора. Можно предоставить пользовательскую функцию фильтрации, которая вызывается характерным для jQuery UI способом, как показано в листинге 31.6.

Листинг 31.6. Применение пользовательской функции фильтрации списка

```
...
<script type="text/javascript">
    $(document).bind("pageinit", function() {
        $('ul').listview("option", "filterCallback",
            function(listItem, filter) {
            var pattern = new RegExp("^" + filter, "i");
            return !pattern.test(listItem)
        })
    })
</script>
...
```

Здесь `listview` — это виджет списка jQuery Mobile, а пользовательская функция фильтрации задается путем вызова для него метода `option()` с использованием данной функции в качестве значения параметра `filterCallback`. Аргументами функции являются текстовое содержимое элемента списка и значение фильтра, введенное пользователем. Функция вызывается однократно для каждого элемента списка, и если она возвращает значение `true`, то элемент, для которого она была вызвана, скрывается. В примере с помощью регулярных выражений отбираются лишь те элементы, которые начинаются с текста, указанного для фильтра. Результат, который соответствует вводу буквы `p` в поле поиска и отображению лишь элемента `Розы`, представлен на рис. 31.8.

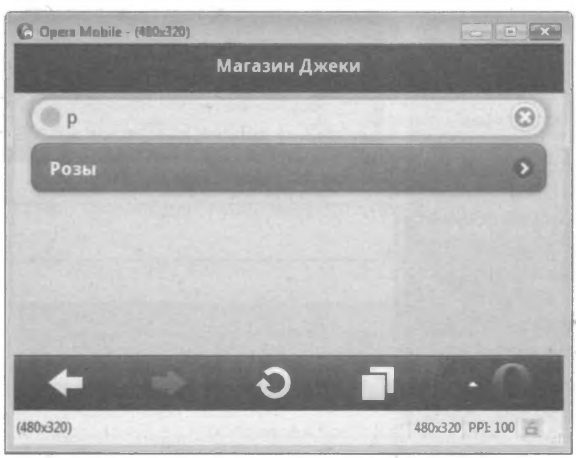


Рис. 31.8. Применение пользовательского фильтра

Форматирование элементов списка

В jQuery Mobile определен ряд атрибутов данных, которые можно использовать для форматирования элементов списка. Эти атрибуты описаны в табл. 31.3.

Работа этих атрибутов демонстрируется в листинге 31.7. Наибольший интерес представляет использование атрибута `data-role` и его значения `list-divider`, в результате чего в списке создается статическая запись, не выступающая в роли навигационного средства. Это полезный механизм, который позволяет добавлять в

список структуру, предоставляющую пользователю контекстную информацию о содержимом списка. В данном примере в список введены дополнительные виды цветов, записи отсортированы в алфавитном порядке и для каждой буквы алфавита, представляющей первую букву каждой записи, добавлены разделители.

Таблица 31.3. Атрибуты для форматирования элементов списков

Атрибут	*Описание*
`data-role`	Если для этого атрибута установлено значение `list-divider`, то элемент списка будет служить разделителем
`data-icon`	Определяет значок для элемента списка с использованием того же набора значений, что и в случае одноименного атрибута виджета атрибута (см. подробное описание в главе 29)
`data-theme`	Применяет указанную палитру к элементу списка
`data-divider-theme`	Применяет указанную палитру к разделителю (применяется к элементам `ul` и `ol`)

Листинг 31.7. Использование разделителей

```
...
<div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Магазин Джеки</h1>
    </div>

    <div data-role="content">
        <ul data-role="listview" data-inset=true data-theme="c"
            data-divider-theme="b">

            <li data-role="list-divider">A</li>
            <li><a href="#astors">Астры</a></li>
            <li data-role="list-divider">Г</li>
            <li><a href="document2.html">Гвоздики</a></li>
            <li data-role="list-divider">Л</li>
            <li><a href="document2.html">Лилии</a></li>
            <li data-role="list-divider">Н</li>
            <li><a href="document2.html">Нарциссы</a></li>
            <li data-role="list-divider">О</li>
            <li><a href="#orchids">Орхидеи</a></li>
            <li data-role="list-divider">П</li>
            <li><a href="document2.html">Пионы</a></li>
            <li><a href="document2.html">Подснежники</a></li>
            <li><a href="document2.html">Примулы</a></li>
            <li data-role="list-divider">Р</li>
            <li><a href="#roses">Розы</a></li>
        </ul>
    </div>
</div>
...
```

Результат введения разделителей в список продемонстрирован на рис. 31.9. Здесь палитра, применяемая ко всему списку, устанавливается с помощью атрибута `data-theme`, а другая палитра, применяемая только к разделителям, устанавливается с помощью атрибута `data-divider-theme`.

Совет. При желании можно придать другой внешний вид лишь отдельным элементам списка, применяя атрибут `data-theme` непосредственно к ним.

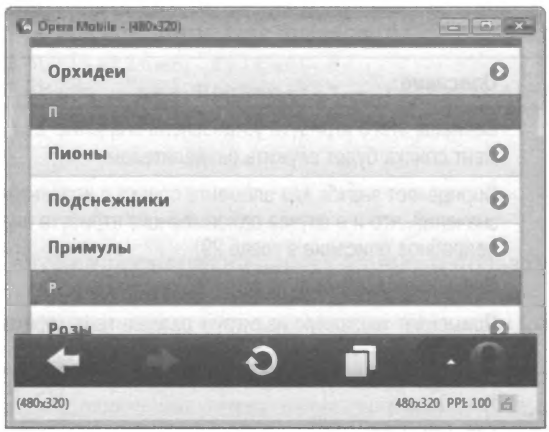


Рис. 31.9. Добавление разделителей в список

Форматирование на основе соглашений

Некоторые опции стилевого оформления управляются правилами, а не конфигурационными параметрами. С примером этого вы уже сталкивались, когда мы рассматривали разделенные списки. Так, при добавлении второго элемента в содержимое элемента `li` библиотека jQuery Mobile автоматически создает элемент разделенного списка. Для получения этого результата применять какие-либо атрибуты данных к элементу списка не требуется — все происходит само собой. В этом разделе вы познакомитесь с тремя различными типами правил, касающихся облачных счетчиков, логического выделения и боковых врезок.

Добавление счетчиков

В элементы списка можно добавлять небольшие числовые индикаторы. Эти индикаторы, так называемые *облачные счетчики* (count bubbles), могут быть полезными в тех случаях, когда элементам списка соответствуют некоторые категории и вы хотите предоставить информацию о количестве объектов в каждой из них. Так, если элементы списка представляют папки электронной почты, то облачные счетчики могут отображать количество сообщений, хранящихся в папках. В приложениях электронной коммерции счетчики могут использоваться для отображения количества имеющихся единиц товара.

Как правило, этой возможностью пользуются для представления числовых значений, но облачные счетчики могут отображать информацию любого рода. Смысл этой информации должен быть очевидным для пользователя ввиду отсутствия в счетчике места для размещения дополнительных пояснений.

Облачный счетчик создается путем включения дополнительного дочернего элемента в содержимое элемента `li`. Этот родительский элемент должен содержать требуемое значение, и ему должен быть назначен класс `ui-li-count`. Пример определения счетчиков, в том числе счетчика с нечисловым значением, приведен в листинге 31.8.

Листинг 31.8. Добавление облачных счетчиков в элементы списка

```
...
<div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Магазин Джеки</h1>
    </div>

    <div data-role="content">
        <ul data-role="listview" data-inset=true data-filter=true
            data-filter-placeholder="Фильтрация списка...">
            <li><a href="#roses">Розы
                <div class="ui-li-count">23</div></a></li>
            <li><div class="ui-li-count">7</div>
                <a href="#orchids">Орхидеи</a></li>
            <li><a href="#astors">Астры</a>
                <div class="ui-li-count">Розовые</div></li>
        </ul>
    </div>

</div>
...
```

Обратите внимание на т, что упомянутый дочерний элемент может располагаться в любом месте внутри элемента `li` и не обязан быть последним (хотя это и является общепринятой практикой). Пример использования облачных счетчиков проиллюстрирован на рис. 31.10.

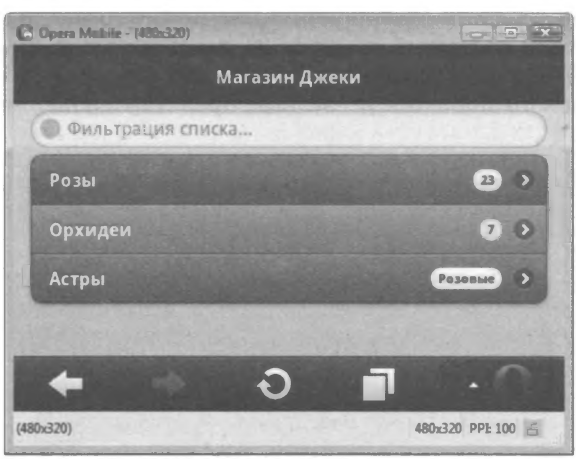


Рис. 31.10. Использование облачных счетчиков

Добавление логического выделения

Если вы используете содержимое, "оберткой" которого служат не элементы `p` (обозначающие абзацы), а заголовочные элементы (от `h1` до `h6`), то jQuery Mobile будет использовать для них логическое выделение различной степени. Это позволяет создать элемент списка, содержащий заголовок и вспомогательный поясняющий текст, как показано в листинге 31.9.

Листинг 31.9. Добавление логического выделения

```
...
<div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Магазин Джеки</h1>
    </div>

    <div data-role="content">
        <ul data-role="listview" data-inset=true data-filter=true
            data-filter-placeholder="Фильтрация списка...">
            <li>
              <a href="#roses"><h1>Розы</h1>
                  <p>Роза - многолетнее древовидное растение
                     семейства розоцветных.</p>
                  <div class="ui-li-count">$4.99</div></a>
            </li>
            <li><div class="ui-li-count">7</div>
                <a href="#orchids">Орхидеи</a></li>
            <li><a href="#astors">Астры</a>
                <div class="ui-li-count">Розовые</div></li>
        </ul>
    </div>

</div>
...
```

В этом примере элемент `h1` используется для вывода названия продукта, а элемент `p` — для вывода поясняющей информации. Кроме того, для отображения цены продукта используется облачный счетчик. (Цена идеально подходит для отображения в счетчиках, поскольку наличие символа валюты делает очевидным смысл приведенного числового значения.) Результат представлен на рис. 31.11.

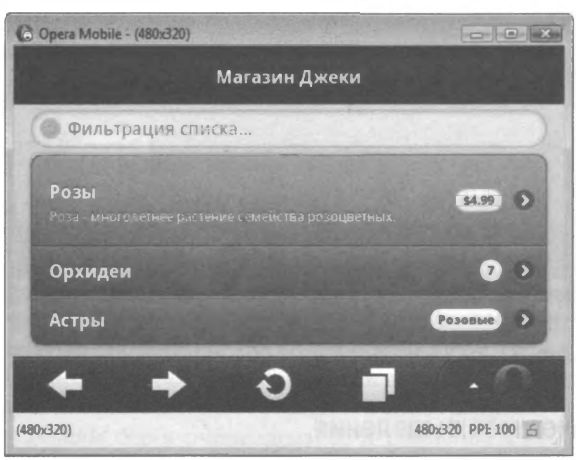


Рис. 31.11. Использование логического выделения в элементах списка

Добавление боковой врезки

Боковая врезка (aside) — это альтернатива использованию облачных счетчиков. Чтобы создать боковую врезку, вы добавляете в элемент `li` дочерний элемент, содержащий информацию, которую нужно отобразить, и назначаете ему класс `ui-li-aside`. Пример использования боковой врезки приведен в листинге 31.10.

Листинг 31.10. Создание боковой врезки в элементе списка

```
...
<div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Магазин Джеки</h1>
    </div>

    <div data-role="content">
        <ul data-role="listview" data-inset=true data-filter=true
            data-filter-placeholder="Фильтрация списка...">
            <li>
              <a href="#roses">
                  <h1>Розы</h1>
                  <p>Роза - многолетнее древовидное растение
                     семейства розоцветных.</p>
                  <p class="ui-li-aside">(Розовая)
                      <strong>$4.99</strong></p>
              </a></li>
            <li><div class="ui-li-count">7</div>
                <a href="#orchids">Орхидеи</a></li>
            <li><a href="#astors">Астры</a>
                <div class="ui-li-count">Розовые</div></li>
        </ul>
    </div>

</div>
...
```

На рис. 31.12 показана боковая врезка для элемента `Розы`.

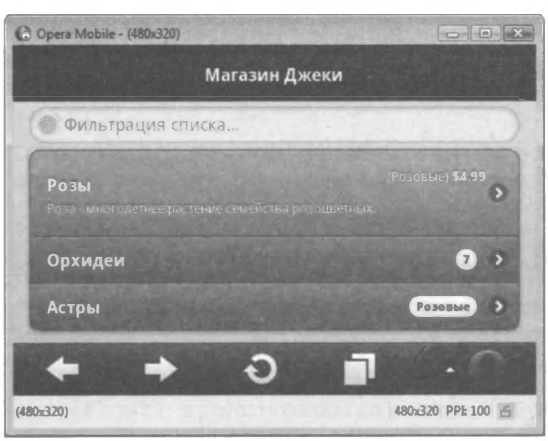


Рис. 31.12. Использование боковой врезки

Резюме

В этой главе был описан виджет списка jQuery Mobile, который может играть роль важного навигационного средства в мобильных приложениях. Были продемонстрированы различные виды списков, стили их представления для пользователя, а также конфигурационные параметры и соглашения, с помощью которых можно управлять содержимым отдельных элементов списка.

ГЛАВА 32

Рефакторинг примера мобильного приложения (часть IV)

В предыдущих главах вы познакомились с библиотекой jQuery Mobile. В этой главе будет реализован более полный пример документа, использующего функциональность jQuery Mobile. По самой своей природе jQuery Mobile гораздо проще библиотеки jQuery UI, следствием чего является меньшее число доступных вариантов выбора проектных решений. Кроме того, разработка с помощью jQuery Mobile подвержена ограничениям, обусловленным специфическими проблемами приложений для мобильных устройств.

Начинаем с простого

В главе 31 приводился пример использования разделенных списков. Этот пример будет задействован в данной главе в качестве отправной точки. Далее в него будет добавлена дополнительная функциональность. Содержимое исходного документа представлено в листинге 32.1.

Листинг 32.1. Исходный документ для данной главы

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        .lcontainer {float: left; text-align: center;
            padding-top: 10px}
        .productData {float: right; padding: 10px; width: 60%}
        .cWrapper {text-align: center; margin: 20px}
    </style>
</head>
```

```
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>

        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true>
                <li><a href="#basket" class="buy"
                        id="rose">Розы</a>
                    <a href="#roses">Розы</a></li>
                <li><a href="#basket" class="buy"
                        id="orchid">Орхидеи</a>
                    <a href="#orchids">Орхидеи</a></li>
                <li><a href="#basket" class="buy"
                        id="astor">Астры</a>
                    <a href="#astors">Астры</a></li>
            </ul>
        </div>
    </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div class="cWrapper">
            Здесь будет находиться корзина
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back"
                data-role="button" data-inline=true
                data-direction="reverse">Назад</a>
        </div>
    </div>

    <div id="roses" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Розы</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="rose.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                        data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                Роза - многолетнее древовидное растение семейства
                розоцветных, произрастающее в виде кустов с
                вертикальными стеблями и вьющимися или плетистыми
                побегами.
                <div><b>Цена: $4.99</b></div>
            </div>
        </div>
    </div>

   <div id="orchids" data-role="page" data-theme="b">
        <div data-role="header">
```

```
            <h1>Орхидеи</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="orchid.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                        data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                Орхидные - многообразное и широко распространенное
                семейство порядка спаржецветных. Это одно из самых
                крупных семейств цветущих растений.
                <div><b>Цена: $10.99</b></div>
            </div>
        </div>
    </div>

    <div id="astors" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Астры</h1>
        </div>
        <div>
            <div class="lcontainer">
                <img src="astor.png">
                <div><a href="#" data-rel="back"
                    data-role="button" data-inline=true
                    data-direction="reverse">Назад</a>
                </div>
            </div>
            <div class="productData">
                  Название "астра" происходит от древнегреческого
                  слова, означающего "звезда", и связано с формой
                  соцветия.
                <div><b>Цена: $2.99</b></div>
            </div>
        </div>
    </div>
</body>
</html>
```

Вставка информации о продуктах программным способом

Первое, что мы сделаем, — заменим статические страницы, описывающие каждый вид цветов, динамически создаваемыми. Благодаря этому мы будем иметь более компактный документ и сможем легко пополнять список цветов, доступных пользователю для выбора, не дублируя HTML-элементы. Страницы будут генерироваться с помощью подключаемого модуля шаблона данных, описанного в главе 12. Этот модуль работает совместно с ядром библиотеки jQuery и поэтому прекрасно вписывается в приложения jQuery Mobile. Я создал файл под названием `data.json`, содержащий данные, необходимые для описания цветов. Содержимое файла `data.json` приведено в листинге 32.2.

Листинг 32.2[1]. Содержимое файла `data.json`

```
[{  "name": "astor",
    "label": "Астры",
    "price": "$2.99",
    "text": "Название 'астра' происходит от древнегреческого
             слова, означающего 'звезда'..."
},{ "name": "carnation",
    "label": "Гвоздики",
    "price": "$1.99",
    "text": "Для выращивания гвоздик подходит влажная нейтральная
             или слегка щелочная почва..."
},{ "name": "daffodil",
    "label": "Нарциссы",
    "price": "$1.99",
    "text": "Нарциссы — это травы, снабженные плотными луковицами
             и лентообразными листьями..."
},{ "name": "rose",
    "label": "Розы",
    "price": "$4.99",
    "text":  "Роза - многолетнее древовидное растение семейства
              розоцветных..."
},{ "name": "orchid",
    "label": "Орхидеи",
    "price": "$10.99",
    "text": "Орхидные - многообразное и широко распространенное
             семейство порядка..."
}]
```

Эти данные описывают пять видов цветов. Для каждого из них я определил название продукта, ярлык, предоставляющий название продукта, которое будет отображаться для пользователя, цену за одну товарную единицу и текстовое описание. Текстовые описания приведены в листинге не полностью, но они содержатся в файле `data.json`, который входит в архив примеров, доступный на сайте книги (см. главу 1).

Теперь, когда у нас имеются все необходимые данные, их можно интегрировать в документ. Изменения, обеспечивающие использование программно генерируемых страниц вместо статических, представлены в листинге 32.3.

Листинг 32.3. Динамическое добавление страниц

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript" src="jquery.tmpl.js"></script>
    <script type="text/javascript">
        $(document).ready(function() {
```

[1] Чтобы избежать проблем с отображением кириллицы, сохраните этот файл в кодировке UTF-8. — *Примеч. ред.*

```
            $.getJSON("data.json", function(data) {
                $('#flowerTmpl').tmpl(data).appendTo("body");
                $('ul').append($('#liTmpl').tmpl(data))
                    .listview("refresh")
            })
        })
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        .lcontainer {float: left; text-align: center;
            padding-top: 10px}
        .productData {float: right; padding: 10px; width: 60%}
        .cWrapper {text-align: center}
    </style>
    <script id="flowerTmpl" type="text/x-jquery-tmpl">
       <div id="${name}" data-role="page" data-theme="b">
            <div data-role="header">
               <h1>${label}</h1>
            </div>
            <div>
                <div class="lcontainer">
                    <img src="${name}.png">
                    <div><a href="#" data-rel="back"
                        data-role="button"
                        data-inline=true
                        data-direction="reverse">Назад</a>
                    </div>
                </div>
                <div class="productData">
                    ${text}
                    <div><b>Цена: ${price}</b></div>
                </div>
            </div>
        </div>
    </script>
    <script id="liTmpl" type="text/x-jquery-tmpl">
        <li>
            <a href="#basket" class="buy" id="${name}">
                ${label}</a>
            <a href="#${name}">${label}</a>
        </li>
    </script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true></ul>
        </div>
    </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
```

```
            <h1>Магазин Джеки</h1>
        </div>
        <div class="cWrapper">
            Здесь будет находиться код корзины
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back"
                data-role="button" data-inline=true
                data-direction="reverse">Назад</a>
        </div>
    </div>
</body>
</html>
```

Я удалил страницы, описывающие отдельные виды цветов, и вместо них использовал подключаемый модуль шаблонов для генерации необходимых элементов на основе данных, получаемых с помощью метода `getJSON()` (описанного в главе 14). Ключевую роль здесь играет следующий простой пользовательский JavaScript-код.

```
<script type="text/javascript">
    $(document).ready(function() {
        $.getJSON("data.json", function(data) {
            $('#flowerTmpl').tmpl(data).appendTo("body");
            $('ul').append($('#liTmpl').tmpl(data))
                .listview("refresh")
        })
    })
</script>
```

Выполнение этого кода отложено до тех пор, пока jQuery не запустит событие `ready`, а не до тех пор, пока не произойдет событие `pageinit` jQuery Mobile. Я хочу, чтобы данные JSON загружались только один раз, а событие `pageinit` происходит многократно, так что привязка к нему нецелесообразна. Получив данные, я вызываю метод `tmpl()` для добавления динамически генерируемых страниц в элемент `body` документа.

Кроме того, для генерации элементов основного списка цветов я использую шаблон. О том, что содержимое списка изменено, я сообщаю jQuery Mobile путем вызова метода `refresh` для виджета списка, как показано ниже.

```
$('ul').append($('#liTmpl').tmpl(data)).listview("refresh")
```

Шаблоны данных довольно просты, и для работы с ними используются стандартные методики, описанные в главе 12. На рис. 32.1 показан полученный результат — список, элементы которого сгенерированы программой, и ссылки на страницы, которые также были добавлены в документ программным способом.

Повторное использование страниц

Мне нравится подход, основанный на шаблонах данных, поскольку он демонстрирует роль jQuery как фундамента многих областей функциональности, что позволяет объединить возможности шаблонов с такими средствами построения интерфейсов, как jQuery Mobile.

Учитывая вышесказанное, можно более применить более элегантный подход для работы со страницами, описывающими отдельные виды цветов. Вместо того чтобы генерировать наборы элементов для каждого цветка в отдельности, можно сгенерировать всего один набор и изменять его для отображения информации о

том цветке, который выбрал пользователь. Изменения, которые необходимо внести в документ для реализации этой идеи, представлены в листинге 32.4.

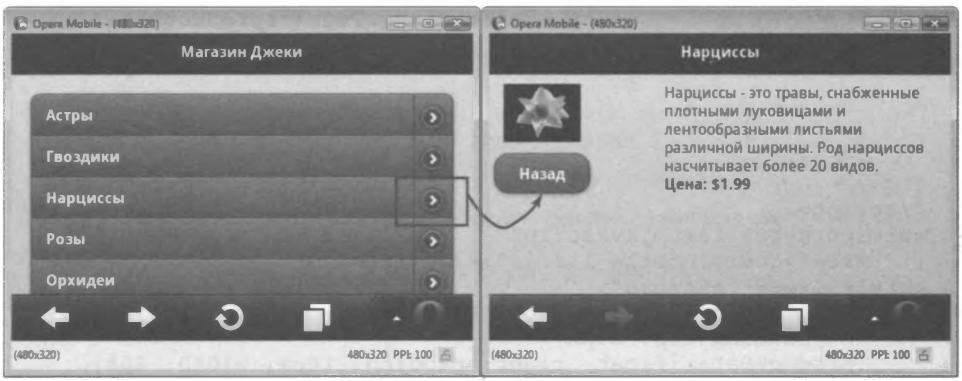


Рис. 32.1. Программно сгенерированные список элементов и страницы

Совет. Этот подход особенно целесообразно использовать в приложениях jQuery Mobile, в которых один HTML-документ может содержать несколько страниц. Как правило, в силу ограничений, свойственных мобильным устройствам, вы заинтересованы в максимальном упрощении своих HTML-документов.

Листинг 32.4. Повторное использование одной страницы для нескольких продуктов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript" src="jquery.tmpl.js"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $.getJSON("data.json", function(data) {
                $('ul').append($('#liTmpl').tmpl(data))
                    .listview("refresh")
                $("a.productLink").bind("tap", function() {
                    var targetFlower = $(this)
                        .attr("data-flower");
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            var page = $('#productPage');
                            page.find("#header").text(data[i]
                                .label);
                            page.find("#image")
                                .attr("src", data[i]
                                .name + ".png");
                            page.find("#description")
                                .text(data[i].text);
```

```
                            page.find("#price").text(data[i]
                                .price);

                            $.mobile.changePage("#productPage");
                            break;
                        }
                    }
                })
            })
        })
    </script>
    <script type="text/javascript"
        src="jquery.mobile-1.0.js"></script>
    <style type="text/css">
        .lcontainer {float: left; text-align: center;
            padding-top: 10px}
        .productData {float: right; padding: 10px; width: 60%}
        .cWrapper {text-align: center}
    </style>
    <script id="liTmpl" type="text/x-jquery-tmpl">
        <li>
            <a href="#basket" class="buy" id="${name}">
                ${label}</a>
            <a class="productLink" data-flower="${name}" href="#">
                ${label}</a>
        </li>
    </script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true>

            </ul>
        </div>
    </div>

    <div id="productPage" data-role="page" data-theme="b">
         <div data-role="header">
            <h1 id="header"></h1>
         </div>
         <div>
             <div class="lcontainer">
                 <img id="image" src="">
                 <div><a href="#" data-rel="back"
                     data-role="button" data-inline=true
                     data-direction="reverse">Назад</a>
                 </div>
             </div>
             <div class="productData">
                 <span id="description"></span>
                 <div><b>Цена: <span id="price"></span></b></div>
             </div>
```

```
        </div>
    </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Магазин Джеки</h1>
        </div>
        <div class="cWrapper">
            Здесь будет находиться код корзины
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back" data-role="button"
                data-inline=true
                data-direction="reverse">Назад</a>
        </div>
    </div>
</body>
</html>
```

Здесь из документа удален один из шаблонов данных и добавлена новая страница (с идентификатором `productPage`), которая используется для всех видов цветов. Я видоизменил шаблон, используемый для генерации элементов списка, таким образом, чтобы в атрибуте `href` отсутствовала ссылка на целевую страницу, и добавил собственный атрибут данных, с помощью которого можно будет узнать, к какому виду цветов относится любая заданная ссылка. После извлечения данных из JSON-файла переработанный вариант сценария выбирает все ссылки на конкретные продукты из списка элементов, который был только что создан с помощью шаблона, и осуществляет привязку к событию `tap`. Когда пользователь касается элемента списка пальцем, сценарий находит соответствующий элемент данных и использует его свойства для конфигурирования страницы `productPage`, задавая при этом текст и изображение, которые должны быть отображены для пользователя, как показано ниже.

```
<script type="text/javascript">
    $(document).ready(function() {
        $.getJSON("data.json", function(data) {
            $('ul').append($('#liTmpl').tmpl(data))
                .listview("refresh")
            $("a.productLink").bind("tap", function() {
                var targetFlower = $(this)
                    .attr("data-flower");
                for (var i = 0; i < data.length; i++) {
                    if (data[i].name == targetFlower) {
                        var page = $('#productPage');
                        page.find("#header").text(data[i]
                            .label);
                        page.find("#image")
                            .attr("src", data[i]
                            .name + ".png");
                        page.find("#description")
                            .text(data[i].text);
                        page.find("#price").text(data[i]
                            .price);

                        $.mobile.changePage("#productPage");
                        break;
```

```
                    }
                }
            })
        })
    })
</script>
```

Сконфигурировав страницу, я организую переход к нужной странице с помощью метода `changePage()`. Внешне документ выглядит в браузере точно так же, как и прежде, но в результате внесения изменений уменьшился набор элементов, которые должен обрабатывать мобильный браузер, и примененный подход наглядно демонстрирует возможности по манипулированию структурой страниц документа jQuery Mobile.

Создание корзины покупателя

В этом примере используется разделенный список, в котором левая составляющая каждого элемента приводит на страницу `basket`. В данном разделе мы определим элементы для страницы и добавим JavaScript-код, реализующий корзину. Изменения, которые для этого требуется внести, представлены в листинге 32.5.

Листинг 32.5. Реализация корзины покупателя

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript" src="jquery.tmpl.js"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $.getJSON("data.json", function(data) {
                $('ul').append($('#liTmpl').tmpl(data))
                    .listview("refresh")
                $("a.productLink").bind("tap", function() {
                    var targetFlower = $(this)
                        .attr("data-flower");
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            var page = $('#productPage');
                            page.find("#header").text(data[i]
                                .label);
                            page.find("#image")
                               .attr("src", data[i]
                               .name + ".png");
                            page.find("#description")
                                .text(data[i].text);
                            page.find("#price").text(data[i]
                                .price);

                            $.mobile.changePage("#productPage");
```

```
                        break;
                    }
                }
            })

            $('a.buy').bind("tap", function() {
                var targetFlower = this.id;
                var row = $("#basketTable tbody #" +
                    targetFlower);
                if (row.length > 0) {
                    var countCell = row.find("#count");
                    countCell.text(Number(countCell.text()) +
                        1);
                } else {
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            $('#trTmpl').tmpl(data[i])
                               .appendTo("#basketTable tbody")
                            break;
                        }
                    }
                }
                calculateTotals();
                $.mobile.changePage("#basket")
            })
        })
    })

    function calculateTotals() {
        var total = 0;
        $('#basketTable tbody').children()
            .each(function(index, elem) {
                var count = Number($(elem)
                    .find("#count").text())
                var price = Number($(elem)
                    .attr("data-price").slice(1))
                var subtotal = count * price;
                $(elem).find("#subtotal").text("$"+
                    subtotal.toFixed(2));
                total += subtotal;
        })
        $('#total').text("$" + total.toFixed(2))
    }
</script>
<script type="text/javascript"
    src="jquery.mobile-1.0.js"></script>
<style type="text/css">
    .lcontainer {float: left; text-align: center;
        padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center}
    table {display: inline-block; margin: auto;
        margin-top: 20px; text-align: left;
        border-collapse: collapse}
    td {min-width: 100px}
```

```
        th, td {text-align: right}
        th:nth-child(1), td:nth-child(1) {text-align: left}
    </style>
    <script id="liTmpl" type="text/x-jquery-tmpl">
        <li>
            <a href="#basket" class="buy" id="${name}">
                ${label}</a>
            <a class="productLink" data-flower="${name}" href="#">
                ${label}</a>
        </li>
    </script>
    <script id="trTmpl" type="text/x-jquery-tmpl">
        <tr data-price="${price}" id="${name}">
            <td>${label}</td><td id="count">1</td>
            <td id="subtotal">0</td></tr>
    </script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true>

            </ul>
        </div>
    </div>

    <div id="productPage" data-role="page" data-theme="b">
         <div data-role="header">
            <h1 id="header"></h1>
         </div>
         <div>
             <div class="lcontainer">
                 <img id="image" src="">
                 <div><a href="#" data-rel="back"
                     data-role="button" data-inline=true
                     data-direction="reverse">Назад</a>
                 </div>
             </div>
             <div class="productData">
                 <span id="description"></span>
                 <div><b>Цена: <span id="price"></span></b></div>
             </div>
         </div>
     </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div class="cWrapper">
            <table id="basketTable" border=0>
                <thead>
                    <tr><th>Цветы</th><th>Количество</th>
                    <th>Всего</th></tr>
```

```
                </thead>
                <tbody></tbody>
                <tfoot>
                    <tr><th colspan=2>ИТОГО:</th><td id="total">
                    </td></tr>
                </tfoot>
            </table>
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back" data-role="button"
                data-inline=true
                data-direction="reverse">Назад</a>
            <button data-inline="true">Заказать</button>
        </div>
    </div>
</body>
</html>
```

Я добавил на страницу `basket` таблицу, отображающую по одной строке для каждого выбранного пользователем продукта. В каждой строке отображается имя продукта, заказанное количество и общая сумма для данного вида цветов. В таблице имеется нижний колонтитул, в котором отображается общая итоговая сумма заказа. Мы привязались к событию `tap`, поэтому щелчок на левой составляющей разделенной кнопки приводит либо к добавлению новой строки в таблицу, либо к увеличению количества заказанных экземпляров цветка, если в таблице уже присутствует строка для данного продукта. Новые строки генерируются с использованием другого шаблона данных, а все остальное обрабатывается путем считывания содержимого элементов документа.

Состояние корзины покупателя определяется и обслуживается средствами DOM. Я мог бы создать объект JavaScript для моделирования заказа и управлять содержимым корзины с помощью этого объекта, но в книге, посвященной jQuery, мне хотелось использовать любую возможность для того, чтобы работать непосредственно с самим документом. В результате мы получаем очень простую корзину, представленную на рис. 32.2.

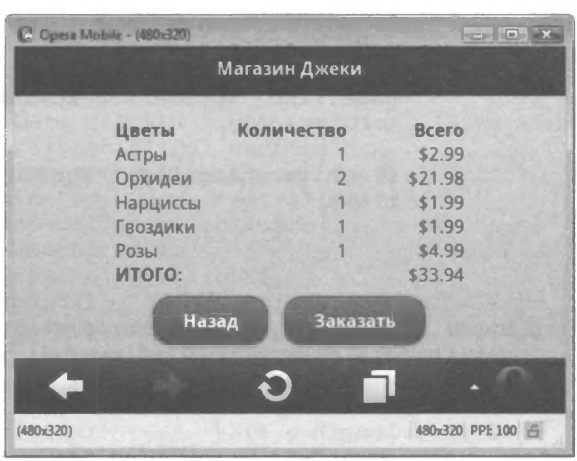


Рис. 32.2. Страница корзины покупателя

Добавление кода для изменения объема заказа

Наша корзина уже вполне работоспособна, но если пользователь хочет заказать, например, две розы, то он должен коснуться элемента Розы, затем коснуться кнопки Назад, а после этого еще раз коснуться элемента Розы. Такая организация процесса выглядит довольно неуклюже, и поэтому, чтобы упростить выбор нужного количества единиц продукта, я добавил некоторые элементы ввода в таблицу. Соответствующие изменения представлены в листинге 32.6.

Листинг 32.6. Добавление полей со счетчиком в таблицу корзины

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript" src="jquery.tmpl.js"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $.getJSON("data.json", function(data) {
                $('ul').append($('#liTmpl').tmpl(data))
                    .listview("refresh")
                $("a.productLink").bind("tap", function() {
                    var targetFlower = $(this)
                        .attr("data-flower");
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            var page = $('#productPage');
                            page.find("#header").text(data[i]
                                .label);
                            page.find("#image")
                               .attr("src", data[i]
                               .name + ".png");
                            page.find("#description")
                                .text(data[i].text);
                            page.find("#price").text(data[i]
                                .price);

                            $.mobile.changePage("#productPage");
                            break;
                        }
                    }
                })

                $('a.buy').bind("tap", function() {
                    var targetFlower = this.id;
                    var row = $("#basketTable tbody #" +
                        targetFlower);
                    if (row.length > 0) {
                        var countCell = row.find("#count input");
                        countCell.val(Number(countCell.val()) +
                            1);
```

```
                } else {
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            $('#trTmpl').tmpl(data[i])
                                .appendTo("#basketTable tbody")
                                .find("input").textinput()
                            break;
                        }
                    }
                }
                calculateTotals();
                $.mobile.changePage("#basket")
            })

            $('input').live("change click", function(event) {
                calculateTotals();
            })
        })
    })

    function calculateTotals() {
        var total = 0;
        $('#basketTable tbody').children()
        .each(function(index, elem) {
            var count = Number($(elem)
                .find("#count input").val())
            var price = Number($(elem)
                .attr("data-price").slice(1))
            var subtotal = count * price;
            $(elem).find("#subtotal")
                .text("$"+ subtotal.toFixed(2));
            total += subtotal;
        })
        $('#total').text("$" + total.toFixed(2))
    }
</script>
<script type="text/javascript"
    src="jquery.mobile-1.0.js"></script>
<style type="text/css">
    .lcontainer {float: left; text-align: center;
        padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center}
    table {display: inline-block; margin: auto;
        margin-top: 20px; text-align: left;
        border-collapse: collapse}
    td {min-width: 100px}
    th, td {text-align: right}
    th:nth-child(1), td:nth-child(1) {text-align: left}
    input[type=number] {background-color: white}
    tfoot tr {border-top: medium solid black}
    tfoot tr td {padding-top: 10px}
</style>
<script id="liTmpl" type="text/x-jquery-tmpl">
    <li>
        <a href="#basket" class="buy" id="${name}">
            ${label}</a>
```

```
            <a class="productLink" data-flower="${name}" href="#">
                ${label}</a>
        </li>
    </script>
    <script id="trTmpl" type="text/x-jquery-tmpl">
        <tr data-theme="b" data-price="${price}"
            id="${name}"><td>${label}</td>
            <td id="count">
                <input type=number value=1 min=0 max=10></td>
            <td id="subtotal">0</td>
        </tr>
    </script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true>

            </ul>
        </div>
    </div>

    <div id="productPage" data-role="page" data-theme="b">
         <div data-role="header">
            <h1 id="header"></h1>
         </div>
         <div>
             <div class="lcontainer">
                 <img id="image" src="">
                 <div><a href="#" data-rel="back"
                     data-role="button" data-inline=true
                     data-direction="reverse">Назад</a>
                 </div>
             </div>
             <div class="productData">
                 <span id="description"></span>
                 <div><b>Цена: <span id="price"></span></b></div>
             </div>
         </div>
     </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div class="cWrapper">
            <table id="basketTable" border=0>
                <thead>
                    <tr><th>Цветы</th><th>Количество</th>
                    <th>Всего</th></tr>
                </thead>
                <tbody></tbody>
                <tfoot>
                    <tr><th colspan=2>ИТОГО:</th><td id="total">
                    </td></tr>
```

```
                </tfoot>
            </table>
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back" data-role="button"
                data-inline=true
                data-direction="reverse">Назад</a>
            <button data-inline="true">Заказать</button>
        </div>
    </div>
</body>
</html>
```

Здесь в элемент td, который входит в шаблон, используемый для генерации рядов таблицы, и предназначен для отображения количества заказанных товарных единиц, вставлен элемент input. Поскольку для этого элемента указан тип number, то в некоторых браузерах рядом с текстовым полем будет появляться пара небольших кнопок с изображениями стрелок, направленных вверх и вниз. Эти кнопки слишком малы для того, чтобы ими можно было манипулировать с помощью касаний, но задание указанного типа гарантирует, что браузер не допустит ввода данных, не соответствующих числам. Для учебных целей, преследуемых в данной главе, такой подход вполне приемлем, однако для реальных проектов он далеко не идеален, поскольку числовые поля поддерживают ввод чисел с плавающей точкой, а это означает, что допустимыми будут признаваться также дробные числа, если их ошибочно введет пользователь.

Когда элементы input добавляются в документ уже после того, как страницы были улучшены средствами jQuery Mobile, я вызываю метод textinput().

```
$('#trTmpl').tmpl(data[i]).appendTo("#basketTable tbody")
    .find("input").textinput()
```

Если не добавить этот вызов, то браузер будет отображать элементы input собственной реализации, а не улучшенные. Вызов метода textinput() заставляет jQuery Mobile улучшить этот элемент, однако подходящая палитра ему при этом не назначается. В связи с этим для элемента input определяется стиль, который устанавливает цвет фона, гармонирующий со стилем оформления других элементов.

```
input[type=number] {background-color: white}
```

Теперь, когда пользователь имеет возможность изменить количество заказанных единиц продукта на странице корзины, значения суммы по отдельным видам цветов, а также итоговую сумму заказа приходится вычислять чаще. Поскольку элементы input добавляются в документ на протяжении всего жизненного цикла приложения, то для обработки событий используется метод live(). Этот метод описан в главе 9. Код обработчика событий приведен ниже.

```
$('input').live("change click", function(event) {
    calculateTotals();
```

Метод live() связывает функцию-обработчик как с событием change, так и с событием click. Браузеры, добавляющие кнопки со стрелками "вверх" и "вниз" в числовые элементы input, после нажатия этих кнопок генерируют событие click, и поэтому наряду с обработкой более вероятного события change мы должны быть готовы к обработке также этого события. Когда происходит любое из указанных двух событий, функция-обработчик просто вызывает функцию calculateTotals(). Корзина покупателя показана на рис. 32.3.

Рис. 32.3. Добавление элементов ввода на страницу корзины покупателя

Добавление кнопки на информационную страницу

Информация о продукте содержит описание цветов, выбранных пользователем, но у пользователя отсутствует возможность добавить ее в корзину. Чтобы несколько расширить функциональность простой корзины, я добавил на страницу продукта кнопку, которая позволяет добавлять в корзину эти элементы. Изменения, которые необходимо для этого внести, представлены в листинге 32.7.

Листинг 32.7. Добавление кнопки на страницу продукта

```
...
<div id="productPage" data-role="page" data-theme="b">
    <div data-role="header">
        <h1 id="header"></h1>
    </div>
    <div>
        <div class="lcontainer">
            <img id="image" src="">
            <div><a href="#" data-rel="back"
                data-role="button" data-inline=true
                data-direction="reverse">Назад</a>
            </div>
        </div>
        <div class="productData">
            <span id="description"></span>
            <div>
                <b>Цена: <span id="price"></span></b>
                <a href="#" id="buybutton" data-flower=""
                    data-role="button"
                    data-inline=true>Купить</a>
            </div>
        </div>
    </div>
</div>
...
```

Здесь определен элемент a, который jQuery Mobile преобразует в виджет кнопки. Атрибут данных data-flower добавлен для того, чтобы можно было отслеживать, какой именно цветок отображается, когда пользователь касается кнопки. Для поддержки этой кнопки в сценарий был введен дополнительный код. Соответствующие изменения представлены в листинге 32.8.

Листинг 32.8. Изменение сценария для поддержки кнопки Купить

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $.getJSON("data.json", function(data) {
            $('ul').append($('#liTmpl').tmpl(data))
                .listview("refresh")
            $("a.productLink").bind("tap", function() {
                var targetFlower = $(this)
                    .attr("data-flower");
                for (var i = 0; i < data.length; i++) {
                    if (data[i].name == targetFlower) {
                        var page = $('#productPage');
                        page.find("#header").text(data[i]
                            .label);
                        page.find("#image")
                            .attr("src", data[i]
                            .name + ".png");
                        page.find("#description")
                            .text(data[i].text);
                        page.find("#price").text(data[i]
                            .price);
                        page.find("#buybutton")
                            .attr("data-flower", data[i].name);

                         $.mobile.changePage("#productPage");
                         break;
                    }
                }
            })

            $('#buybutton').bind("tap", function() {
                addProduct($(this).attr("data-flower"));
            })

            $('a.buy').bind("tap", function() {
                addProduct(this.id);
            })

            function addProduct(targetFlower) {
                var row = $("#basketTable tbody #" +
                    targetFlower);
                if (row.length > 0) {
                    var countCell = row.find("#count input");
                    countCell.val(Number(countCell.val()) + 1);
                } else {
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
```

```
                        $('#trTmpl').tmpl(data[i])
                            .appendTo("#basketTable tbody")
                            .find("input").textinput()

                        break;
                    }
                }
            }
            calculateTotals();
            $.mobile.changePage("#basket")
        }

        $('input').live("change click", function(event) {
            calculateTotals();
        })
    })

    function calculateTotals() {
        var total = 0;
        $('#basketTable tbody').children()
        .each(function(index, elem) {
            var count = Number($(elem)
                .find("#count input").val())
            var price = Number($(elem)
                .attr("data-price").slice(1))
            var subtotal = count * price;
            $(elem).find("#subtotal")
                .text("$"+ subtotal.toFixed(2));
            total += subtotal;
        })
        $('#total').text("$" + total.toFixed(2))
    }
</script>
...
```

Эти изменения довольно очевидны. При выборе пользователем продукта в основном списке для атрибута `data-flower` элемента `a` устанавливается соответствующее значение. Для кнопки регистрируется обработчик события `tap`, а значение атрибута `data-flower` используется для вызова функции `addProduct()`, которая содержит код, извлеченный из другой функции-обработчика. В результате этих изменений пользователь может добавлять продукты в корзину из основного списка (коснувшись левой части элемента разделенного списка) или из информационной страницы (коснувшись кнопки `Купить`). Результат добавления кнопки `Купить` на информационную страницу представлен на рис. 32.4.

Реализация процедуры завершения заказа

В завершение я продемонстрирую, как организовать сбор данных с различных страниц jQuery Mobile с помощью формы, которая может быть использована для выполнения Ajax-запросов, хотя ни выполнение самих запросов, ни реализация сервера в мои планы не входит. Библиотека jQuery Mobile использует описанную в главах 14 и 15 поддержку Ajax, предоставляемую ядром библиотеки jQuery. Доработанный вариант примера, в который добавлены страница, отображаемая для

пользователя при касании кнопки `Заказать`, и функция-обработчик событий, осуществляющая сбор данных, представлены в листинге 32.9.

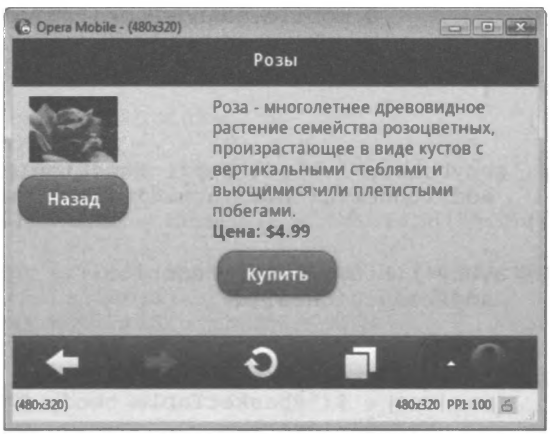


Рис. 32.4. Добавление кнопки на информационную страницу продукта

Листинг 32.9. Реализация процедуры завершения заказа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <meta name="viewport" content="width=device-width,
        initial-scale=1">
    <link rel="stylesheet"
        href="jquery.mobile-1.0.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.6.4.js"></script>
    <script type="text/javascript" src="jquery.tmpl.js"></script>
    <script type="text/javascript">

        $(document).ready(function() {
            $.getJSON("data.json", function(data) {
                $('ul').append($('#liTmpl').tmpl(data))
                    .listview("refresh")
                $("a.productLink").bind("tap", function() {
                    var targetFlower = $(this)
                        .attr("data-flower");
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            var page = $('#productPage');
                            page.find("#header").text(data[i]
                                .label);
                            page.find("#image")
                                .attr("src", data[i]
                                .name + ".png");
                            page.find("#description")
                                .text(data[i].text);
                            page.find("#price").text(data[i]
                                .price);
```

```
                page.find("#buybutton")
                    .attr("data-flower", data[i].name);

                $.mobile.changePage("#productPage");
                break;
            }
        }
    })

    $('#buybutton').bind("tap", function() {
        addProduct($(this).attr("data-flower"));
    })

    $('a.buy').bind("tap", function() {
        addProduct(this.id);
    })

    function addProduct(targetFlower) {
        var row = $("#basketTable tbody #" +
            targetFlower);
        if (row.length > 0) {
            var countCell = row.find("#count input");
            countCell.val(Number(countCell.val()) +
                1);
        } else {
            for (var i = 0; i < data.length; i++) {
                if (data[i].name == targetFlower) {
                    $('#trTmpl')
                        .tmpl(data[i])
                        .appendTo("#basketTable tbody")
                        .find("input").textinput()

                    break;
                }
            }
        }
        calculateTotals();
        $.mobile.changePage("#basket")
    }

    $('input').live("change click", function(event) {
        calculateTotals();
    })

    $('#submit').bind("tap", function() {
        var dataObject = new Object();
        $('#basketTable tbody').children()
            .each(function(index, elem) {
                dataObject[elem.id] = $(elem)
                .find("#count input").val();
        })
        dataObject["name"] = $('#name').val();
        dataObject["wrap"] = $('option:selected')
            .val();
        dataObject["shipping"] = $('input:checked')
            .attr("id")
```

```
                console.log("DATA: " + JSON
                    .stringify(dataObject))
            })

        })
    })

    function calculateTotals() {
        var total = 0;
        $('#basketTable tbody').children()
            .each(function(index, elem) {
            var count = Number($(elem).find("#count input")
                .val())
            var price = Number($(elem).attr("data-price")
                .slice(1))
            var subtotal = count * price;
            $(elem).find("#subtotal").text("$"+ subtotal
                .toFixed(2));
            total += subtotal;
        })
        $('#total').text("$" + total.toFixed(2))
    }

</script>
<script type="text/javascript"
    src="jquery.mobile-1.0.js"></script>
<style type="text/css">
    .lcontainer {float: left; text-align: center;
        padding: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center}
    table {display: inline-block; margin: auto;
        margin-top: 20px; text-align: left;
       border-collapse: collapse}
    td {min-width: 100px; padding-bottom: 10px}
    td:nth-child(2) {min-width: 75px; width: 75px}
    th, td {text-align: right}
    th:nth-child(1), td:nth-child(1) {text-align: left}
    input[type=number] {background-color: white}
    tfoot tr {border-top: medium solid black}
    tfoot tr td {padding-top: 10px}
</style>
<script id="liTmpl" type="text/x-jquery-tmpl">
    <li>
        <a href="#" class="buy" id="${name}">${label}</a>
        <a class="productLink" data-flower="${name}"
            href="#">${label}</a>
    </li>
</script>
<script id="trTmpl" type="text/x-jquery-tmpl">
    <tr data-theme="b" data-price="${price}"
            id="${name}"><td>${label}</td>
        <td id="count"><input type=number value=1
            min=0 max=10></td>
        <td id="subtotal">0</td>
    </tr>
</script>
```

```
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true></ul>
        </div>
    </div>

    <div id="productPage" data-role="page" data-theme="b">
         <div data-role="header">
            <h1 id="header"></h1>
         </div>
         <div>
             <div class="lcontainer">
                 <img id="image" src="">
                 <div><a href="#" data-rel="back"
                     data-role="button" data-inline=true
                     data-direction="reverse">Назад</a>
                 </div>
             </div>
             <div class="productData">
                 <span id="description"></span>
                 <div>
                    <b>Цена: <span id="price"></span></b>
                    <a href="#" id="buybutton" data-flower=""
                        data-role="button"
                        data-inline=true>Buy</a>
                 </div>
             </div>
         </div>
     </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div class="cWrapper">
            <table id="basketTable" border=0>
                <thead>
                    <tr><th>Цветы</th><th>Количество</th>
                        <th>Всего</th></tr>
                </thead>
                <tbody></tbody>
                <tfoot>
                    <tr><th colspan=2>ИТОГО:</th><td id="total">
                    </td></tr>
                </tfoot>
            </table>
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back" data-role="button"
                data-inline=true
                data-direction="reverse">Назад</a>
            <a href="#checkout" data-role="button"
                data-inline="true">Заказать</a>
```

```
        </div>
    </div>

    <div id="checkout" data-role="page" data-theme="b">
        <div data-role="header">
           <h1>Магазин Джеки</h1>
        </div>
        <div data-role="content">

            <label for="name">Имя: </label>
            <input id="name" placeholder="Ваше имя">

            <label for="wrap"><span>Подарочная упаковка:
                </span></label>
            <select id="wrap" name="wrap" data-role="slider">
                <option value="yes" selected>Да</option>
                <option value="no">Нет</option>
            </select>

            <fieldset data-role="controlgroup">
                <legend>Shipping:</legend>
                <input type="radio" name="ship" id="overnight"
                    checked />
                <label for="overnight">В течение суток</label>
                <input type="radio" name="ship" id="23day"/>
                <label for="23day">2-3 days</label>
                <input type="radio" name="ship" id="710day"/>
                <label for="710day">7-10 дней</label>
            </fieldset>

            <div class="cWrapper">
                <a href="#" data-rel="back" data-role="button"
                    data-inline="true"
                    data-direction="reverse">Назад</a>
                <a href="#" id="submit" data-role="button"
                   data-inline=true">Отправить заказ</a>
            </div>

        </div>
    </div>

</body>
</html>
```

Новая страница называется `checkout`. В форме, которую я сделал очень простой, пользователю предлагается ввести имя и выбрать вариант упаковки и сроки доставки. Внешний вид страницы представлен на рис. 32.5. Для показа страницы использована портретная ориентация, которая позволяет отобразить все элементы, не прокручивая экран.

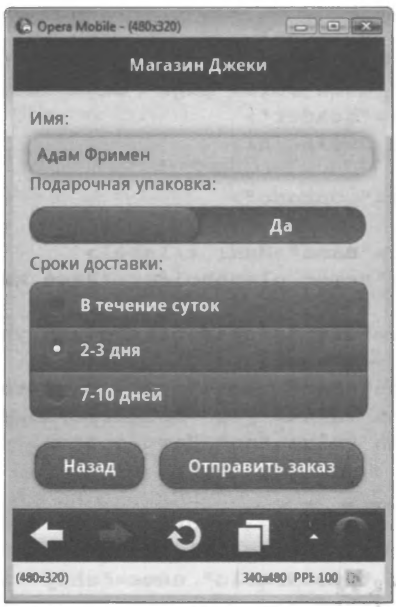


Рис. 32.5. Страница checkout

Касание кнопки `Отправить заказ` инициирует сбор данных с различных страниц HTML-документа и вывод результата на консоль в виде строки в формате JSON. Пример такой строки представлен ниже.

```
{"carnation":"3","rose":"1","orchid":"1",
"name":"Адам Фримен","wrap":"yes","shipping":"23day"}
```

Резюме

В этой главе мы воспользовались основными возможностями библиотеки jQuery Mobile, объединив их для создания простой мобильной реализации примера цветочного магазина. По самой своей природе библиотека jQuery Mobile гораздо проще библиотеки jQuery UI. Главной проблемой является нахождение проектного решения, обеспечивающего предоставление пользователю необходимой информации при небольших размерах экрана.

Часть VI

Дополнительные возможности

ГЛАВА 33

Использование служебных методов jQuery

Библиотека jQuery включает ряд методов, которые либо выполняют операции повышенной сложности над объектами jQuery, либо дополняют язык JavaScript, обеспечивая возможности, которые обычно предоставляются самим языком программирования. Не исключено, что ни один из этих методов вам никогда не понадобится, но библиотека jQuery использует их для собственных нужд, и знакомство с ними поможет сэкономить вам время и усилия, если вы столкнетесь с проблемой, которая уже давно решена командой jQuery.

Одни из этих методов применяются к объектам jQuery, другие вызываются для основной функции jQuery — так называемой функции $ (см. главу 5). Перечень тем, рассматриваемых в данной главе, приведен в табл. 33.1.

Таблица 33.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Помещение операций в очередь для отложенного выполнения	Используйте универсальные очереди	1, 2
Фильтрация содержимого массива	Используйте метод `grep()`	3, 4
Определение того, содержит ли массив указанный объект или значение	Используйте метод `inArray()`	5
Привязка элементов одного массива к элементам другого массива	Используйте метод `map()`	6, 7
Объединение двух массивов	Используйте метод `merge()`	8
Удаление дубликатов из объекта `jQuery` и сортировка оставшихся элементов в том порядке, в каком они встречаются в документе	Используйте метод `unique()`	9
Определение типа объекта	Используйте метод `isXXX()` или `type()`	10, 11
Подготовка содержимого формы к отправке	Используйте метод `serialize()` или `serializeArray()`	12
Синтаксический разбор данных с целью их приведения к более полезному виду	Используйте метод `parseJSON()` или `parseXML()`	13
Удаление ведущих и концевых пробелов из строки	Используйте метод `trim`	14

Окончание табл. 33.1

Задача	Решение	Листинг
Определение того, содержит ли один элемент другой	Используйте метод `contains()`	15
Создание функции-посредника для обработчика событий	Используйте метод `proxy()`	16

Использование универсальных очередей

В главе 10 было продемонстрировано использование очередей для управления цепочкой эффектов jQuery, применяемых к набору элементов. В действительности очередь эффектов — это всего лишь один из возможных типов очередей, тогда как сам механизм очередей универсален и может применяться для самых разных целей. В табл. 33.2 повторно перечислены методы, предназначенные для работы с очередями, несколько видоизмененные для обеспечения универсальности их применения.

Таблица 33.2. Методы для работы с очередями

Метод	Описание
`clearQueue(`*<имя>*`)`	Удаляет из указанной очереди все функции, которые не были запущены к данному моменту
`queue(`*<имя>*`)`	Возвращает указанную очередь функций, которые должны быть выполнены для элементов, содержащихся в объекте `jQuery`
`queue(`*<имя>*`, `*функция*`)`	Добавляет указанную функцию в конец очереди
`dequeue(`*<имя>*`)`	Удаляет из очереди первую из находящихся в ней функций, одновременно выполняя ее для элементов, содержащихся в объекте `jQuery`
`delay(`*<продолжительность>*`, `*<имя>*`)`	Вставляет задержку между эффектами, находящимися в указанной очереди

Если имя очереди не указано, jQuery использует принятое по умолчанию имя `fx`, которому соответствует очередь визуальных эффектов. Для создания очереди функций следует использовать любое другое имя.

Для универсальных очередей jQuery (в отличие от очередей эффектов) вместо метода `stop()` используется метод `clearQueue()`. Из метода `clearQueue()` исключена поддержка очередей эффектов jQuery, предусмотренная в методе `stop()`, как непригодная для более широкого применения. Пример универсальной очереди представлен в листинге 33.1.

Листинг 33.1. Использование очереди

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
```

```
    <script type="text/javascript">
        $(document).ready(function() {

            var elems = $('input');

            elems.queue("gen", function(next) {
                $(this).val(100).css("border", "thin red solid");
                next();
            });

            elems.delay(1000, "gen");

            elems.queue("gen", function(next) {
                $(this).val(0).css("border", "");
                $(this).dequeue("gen");
            });

           $("<button>Обработать очередь</button>")
               .appendTo("#buttonDiv")
               .click(function(e) {
                    elems.dequeue("gen");
                    e.preventDefault();
                });

        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                      <img src="astor.png"/>
                      <label for="astor">Астры:</label>
                      <input name="astor" value="0" required />
                    </div>
                    <div class="dcell">
                      <img src="daffodil.png"/>
                      <label for="daffodil">Нарциссы:</label>
                      <input name="daffodil" value="0" required/>
                    </div>
                    <div class="dcell">
                      <img src="rose.png"/>
                      <label for="rose">Розы:</label>
                      <input name="rose" value="0" required />
                    </div>
                </div>
                <div id="row2"class="drow">
                    <div class="dcell">
                      <img src="peony.png"/>
                      <label for="peony">Пионы:</label>
                      <input name="peony" value="0" required />
                    </div>
                    <div class="dcell">
```

```
                    <img src="primula.png"/>
                    <label for="primula">Примулы:</label>
                    <input name="primula" value="0" required />
                  </div>
                  <div class="dcell">
                    <img src="snowdrop.png"/>
                    <label for="snowdrop">Подснежники:</label>
                    <input name="snowdrop" value="0" required />
                  </div>
              </div>
          </div>
      </div>
      <div id="buttonDiv">
          <button type="submit">Заказать</button>
      </div>
  </form>
</body>
</html>
```

В этом примере создается очередь с именем `gen`, которая воздействует на все элементы `input` в документе и включает три функции. Во-первых, все входные значения устанавливаются равными `100` с помощью метода `val()`, а текстовые поля ввода окружаются рамками с помощью метода `css()`. Во-вторых, с помощью метода `delay()` в очередь добавляется задержка, длительность которой составляет одну секунду. Наконец, в-третьих, мы используем методы `val()` и `css()` для возврата элементов `input` в исходное состояние.

Кроме того, в документ добавляется кнопка, щелчок на которой приводит к вызову метода `dequeue()`. В отличие от очереди эффектов ответственность за запуск обработки с использованием универсальной очереди возлагается на вас. Результат проиллюстрирован на рис. 33.1.

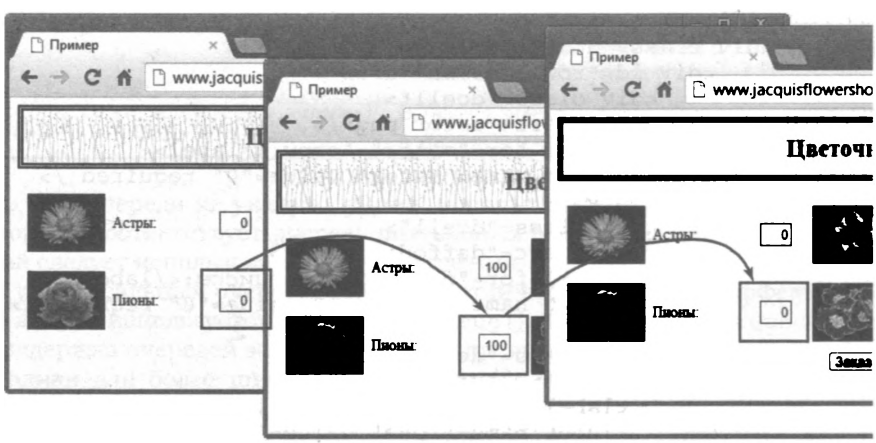


Рис. 33.1. Использование универсальной очереди

Функции, которые помещаются в очередь, работают точно так же, как и в случае очередей событий, и вы сами отвечаете либо за вызов метода `dequeue()`, либо за вызов функции, которая передается очереди в качестве аргумента. Я предпочитаю последний способ, поскольку часто забываю указывать имя очереди при вызове метода `dequeue()`, что нарушает работу очереди.

Обработка элементов очереди вручную

Конечно же, речь вовсе не идет о том, чтобы вы вручную запускали функции, находящиеся в очереди, одну из другой. Во всем, что касается запуска отдельных функций и их исключения из очереди, можно полагаться на внешний триггер, в качестве которого может выступать пользователь, выполняющий щелчок на кнопке, добавленной в документ. Как именно это можно сделать, показано в листинге 33.2.

Листинг 33.2. Принудительное исключение функций

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('input').queue("gen", function() {
            $(this).val(100).css("border", "thin red solid");
        }).queue("gen", function() {
            $(this).val(0).css("border", "");
        }).queue("gen", function() {
            $(this).css("border", "thin blue solid");
            $('#dequeue').attr("disabled", "disabled");
        });

        $("<button id=dequeue>Исключить элемент из очереди
            </button>")
            .appendTo("#buttonDiv")
            .click(function(e) {
                $('input').dequeue("gen");
                e.preventDefault();
            });
    });
</script>
...
```

В этом сценарии я объединил все вызовы метода `queue()` в одну цепочку и добавил функцию, которая создает рамки (устанавливает границы) для выбранных элементов и отключает элемент `button`. Для последовательной обработки элементов очереди нужно каждый раз щелкать на кнопке — автоматизация выполнения цепочки полностью отсутствует. Результаты последовательного выполнения операций цепочки представлены на рис. 33.2.

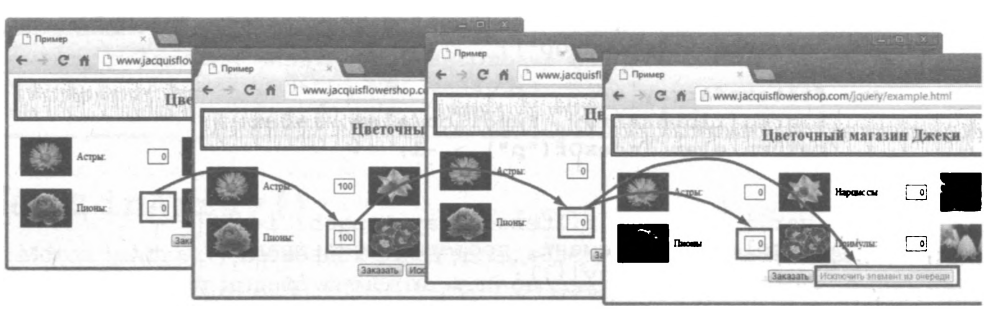


Рис. 33.2. Управление очередью вручную

Служебные методы для работы с массивами

Библиотека jQuery предоставляет ряд полезных методов для работы с массивами. Эти методы описаны в табл. 33.3. В большинстве случаев для работы с массивами HTMLElement существуют лучшие способы — вам достаточно всего лишь воспользоваться стандартными методами jQuery, предназначенными для обработки и фильтрации элементов. Однако в случае других видов массивов эти функции могут вам пригодиться.

Таблица 33.3. Служебные методы для работы с массивами

Метод	*Описание*

`$.grep(`*<массив>*`,` *функция*`)`
`$.grep(`*<массив>*`,` *функция*`,`
логическое_значение_invert`)`
	Выбирает из массива элементы, удовлетворяющие указанной функции фильтра

`$.inArray(`*<значение>*`,` *<массив>*`)`	Определяет, содержится ли в массиве указанное значение
`$.map(`*<массив>*`,` *функция*`)` `$.map(`*<массив>*`,` *<объект>*`)`	Отображает массив или объект в другой массив или объект, используя указанную функцию
`$.merge(`*<массив>*`,` *<массив>*`)`	Присоединяет содержимое второго массива к содержимому первого
`$.unique(HTMLElement[])`	Сортирует массив объектов HTMLElement в том порядке, в котором они встречаются в документе, и удаляет дубликаты

Метод `grep()`

Метод `grep()` позволяет находить все элементы массива, удовлетворяющие функции фильтра. Пример использования этого метода приведен в листинге 33.3.

Листинг 33.3. Использование метода `grep()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var flowerArray = ["astor", "daffodil", "rose", "peony",
            "primula", "snowdrop"];

        var filteredArray =
            $.grep(flowerArray, function(elem, index) {
            return elem.indexOf("p") > -1;
        });

        for (var i = 0; i < filteredArray.length; i++) {
            console.log("Элемент, прошедший фильтрацию: " +
                filteredArray[i]);
        }
    });
</script>
...
```

Функции фильтра передаются два аргумента. Первый из них является элементом массива, второй — индексом этого элемента. Функция вызывается для каждого элемента массива, и в выходной результат включаются лишь те элементы, для которых функция возвращает значение `true`.

В этом примере метод `grep()` применяется к массиву строк. В процессе фильтрации отбрасываются строки, в которых отсутствует буква `p` (латиница). Содержимое отфильтрованного массива выводится на консоль. Результат выглядит следующим образом.

```
Элемент, прошедший фильтрацию: peony
Элемент, прошедший фильтрацию: primula
Элемент, прошедший фильтрацию: snowdrop
```

Методу `grep()` можно предоставить дополнительный аргумент. Если этот аргумент равен `true`, то процесс фильтрации обращается, т.е. в отфильтрованный результат включаются лишь те элементы, для которых функция возвращает `false`. Пример использования этого аргумента приведен в листинге 33.4.

Листинг 33.4. Обращение процесса отбора с использованием метода `grep()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var flowerArray = ["astor", "daffodil", "rose", "peony",
            "primula", "snowdrop"];

        var filteredArray =
            $.grep(flowerArray, function(elem, index) {
            return elem.indexOf("p") > -1;
        }, true);

        for (var i = 0; i < filteredArray.length; i++) {
            console.log("Элемент, прошедший фильтрацию: " +
                filteredArray[i]);
        }
    });
</script>
...
```

В данном случае мы получаем следующий результат.

```
Элемент, прошедший фильтрацию: astor
Элемент, прошедший фильтрацию: daffodil
Элемент, прошедший фильтрацию: rose
```

Метод `inArray()`

Метод `inArray()` позволяет определить, содержит ли массив указанное значение, и возвращает индекс элемента, если он содержится в массиве; в противном случае возвращается `-1`. Пример использования метода `inArray()` приведен в листинге 33.5.

Листинг 33.5. Использование метода `inArray()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var flowerArray = ["astor", "daffodil", "rose", "peony",
            "primula", "snowdrop"];

        console.log("Массив содержит rose: " +
            $.inArray("rose", flowerArray));
        console.log("Массив содержит lily: " +
            $.inArray("lily", flowerArray));
    });
</script>
...
```

В этом сценарии проверяется, содержит ли массив цветов названия `rose` и `lily`. Результат имеет следующий вид.

```
Массив содержит rose: 2
Массив содержит lily: -1
```

Метод `map()`

Метод `map()` позволяет преобразовать содержимое массива или объекта, подобного массиву, в новый массив с помощью функции, которая определяет, каким образом каждый элемент должен быть преобразован. Пример использования метода `map()` в случае массива приведен в листинге 33.6.

Листинг 33.6. Использование метода `map()` для преобразования массива

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var flowerArray = ["astor", "daffodil", "rose", "peony",
            "primula", "snowdrop"];

        var result = $.map(flowerArray, function(elem, index) {
            return index + ": " + elem;
        });

        for (var i = 0; i < result.length; i++) {
            console.log(result[i]);
        }
    });
</script>
...
```

Функция преобразования выполняется для каждого элемента массива, принимая в качестве аргументов элемент и его индекс. Результат, возвращаемый функцией, включается в массив, возвращаемый методом `map()`. В данном сценарии каждый элемент массива преобразуется путем конкатенации значения и индекса элемента, что дает следующий результат.

```
0: astor
1: daffodil
2: rose
3: peony
4: primula
5: snowdrop
```

Метод `map()` можно использовать для избирательного преобразования массива. Если для какого-либо обрабатываемого элемента функция не возвращает никакого значения, то этот элемент в результирующий массив не включается. Пример такого использования метода `map()` приведен в листинге 33.7.

Листинг 33.7. Избирательное преобразование массива

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var flowerArray = ["astor", "daffodil", "rose", "peony",
            "primula", "snowdrop"];

        var result = $.map(flowerArray, function(elem, index) {
            if (elem != "rose") {
                return index + ": " + elem;
            }
        });

        for (var i = 0; i < result.length; i++) {
            console.log(result[i]);
        }
    });
</script>
...
```

Результаты генерируются для всех элементов массива, кроме `rose`, так что в данном случае консольный вывод будет выглядеть следующим образом.

```
0: astor
1: daffodil
3: peony
4: primula
5: snowdrop
```

Метод `merge()`

Метод `merge()` объединяет два массива, как показано в листинге 33.8.

Листинг 33.8. Использование метода `merge()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var flowerArray = ["astor", "daffodil", "rose", "peony",
            "primula", "snowdrop"];
```

```
        var additionalFlowers = ["carnation", "lily", "orchid"];

        $.merge(flowerArray, additionalFlowers);

        for (var i = 0; i < flowerArray.length; i++) {
            console.log(flowerArray[i]);
        }
    });
</script>
...
```

Элементы второго массива объединяются с элементами первого массива и сохраняются в массиве, указанном в качестве первого аргумента. В этом примере выполнение сценария приводит к следующему результату.

```
astor
daffodil
rose
peony
primula
snowdrop
carnation
lily
orchid
```

Метод `unique()`

Метод `unique()` сортирует массив объектов `HTMLElement` в том порядке, в котором они встречаются в документе, и при этом удаляет все дубликаты элементов. Пример использования этого метода приведен в листинге 33.9.

Листинг 33.9. Использование метода `unique()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var selection = $('img[src*=rose], img[src*=primula]')
            .get();
        $.merge(selection, $('img[src*=astor]'));
        $.merge(selection, $('img'));

        $.unique(selection);

        for (var i =0; i < selection.length; i++) {
            console.log("Элемент: " + selection[i].src);
        }

    });
</script>
...
```

Процесс сортировки осуществляется "на месте" в том смысле, что массив, передаваемый методу `unique()` в качестве аргумента, изменяется. В этом примере метод `unique()` применяется к массиву объектов `HTMLElement`, который был специально

создан так, чтобы в нем содержались дубликаты элементов, а порядок элементов не совпадал с порядком их появления в документе.

Служебные методы для работы с типами

Библиотека jQuery предоставляет набор методов, которые могут быть использованы для определения природы объектов JavaScript (табл. 33.4).

Таблица 33.4. Служебные методы для работы с типами

Метод	Описание
`$.isArray(`*объект*`)`	Возвращает `true`, если объект является массивом
`$.isEmptyObject(`*объект*`)`	Возвращает `true`, если объект не содержит ни свойств, ни методов
`$.isFunction(`*объект*`)`	Возвращает `true`, если объект является функцией
`$.isNumeric(`*объект*`)`	Возвращает `true`, если объект является числом
`$.isWindow(`*объект*`)`	Возвращает `true`, если объект является объектом `Window`
`$.isXMLDoc(`*объект*`)`	Возвращает `true`, если объект является XML-документом
`$.type(`*объект*`)`	Возвращает встроенный JavaScript-тип объекта

Большинство этих методов очень просты. Вы передаете объект методу, и тот возвращает значение `true`, если объект относится к тому типу, для тестирования которого данный метод предназначен, и `false` — в противном случае. В качестве простой демонстрации в листинге 33.10 представлен пример использования метода `isFunction()`.

Листинг 33.10. Использование метода `isFunction()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
            function myFunc() {
            console.log("Привет!");
        }
        console.log("IsFunction: " + $.isFunction(myFunc));
        console.log("IsFunction: " + $.isFunction("hello"));
    });
</script>
...
```

В этом сценарии метод `isFunction()` используется для тестирования двух объектов. Результат выглядит следующим образом.

```
IsFunction: true
IsFunction: false
```

Метод `type()`

Метод `type()` имеет то незначительное отличие, что возвращает базовый JavaScript-тип объекта. Результатом может быть одна из следующих строк:

- boolean;
- number;
- string;
- function;
- array;
- date;
- regexp;
- object.

Пример использования метода `type()` приведен в листинге 33.11.

Листинг 33.11. Использование метода `type()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        function myFunc() {
            console.log("Привет!");
        }

        var jq = $('img');
        var elem = document.getElementById("row1");

        console.log("Тип: " + $.type(myFunc));
        console.log("Тип: " + $.type(jq));
        console.log("Тип: " + $.type(elem));
    });
</script>
...
```

В этом сценарии метод `type()` используется для тестирования функции, объекта `jQuery` и объекта `HTMLElement`. Результат выглядит следующим образом.

```
Тип: function
Тип: object
Тип: object
```

Служебные методы для работы с данными

В библиотеке jQuery определен ряд служебных методов, которые могут быть использованы для работы с различными видами данных (табл. 33.5).

Таблица 33.5. Служебные методы для работы с данными

Метод	*Описание*
`serialize()`	Кодирует набор элементов формы в строку, пригодную для отправки на сервер

Окончание табл. 33.5

Метод	Описание
`serializeArray()`	Кодирует набор элементов формы в строку, подготовленную для кодирования в формате JSON
`$.parseJSON(<json>)`	Создает JavaScript-объект из данных JSON
`$.parseXML(<xml>)`	Создает объект `XMLDocument` из строки XML
`$.trim(строка)`	Удаляет все пробельные символы, находящиеся в начале и конце строки

Сериализация данных формы

Методы `serialize()` и `serializeArray()` удобным образом кодируют набор элементов формы в виде строки, пригодной для отправки на сервер как обычным способом, так и посредством Ajax. Пример использования обоих методов приведен в листинге 33.12.

Листинг 33.12. Сериализация данных формы

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Serialize</button>")
            .appendTo("#buttonDiv").click(function(e) {

            var formArray = $('form').serializeArray();
            console.log("JSON: " + JSON.stringify(formArray))

            var formString = $('form').serialize();
            console.log("String: " + formString)

            e.preventDefault();
        });

    });
</script>
...
```

В этом примере для сериализации элементов формы используются оба метода, а результаты выводятся на консоль. Метод `serializeArray()` возвращает массив JavaScript, содержащий по одному объекту для каждого элемента формы в документе. Каждый из этих объектов имеет два свойства: свойство `name` содержит значение атрибута `name` элемента, а свойство `value` — значение элемента. Консольный вывод для этого примера приводится ниже.

```
[{"name":"astor","value":"1"},{"name":"daffodil","value":"0"},
 {"name":"rose","value":"0"},{"name":"peony","value":"0"},
 {"name":"primula","value":"2"},{"name":"snowdrop","value":"0"}]
```

В отличие от этого метод `serialize()` создает кодированную строку.

```
astor=1&daffodil=0&rose=0&peony=0&primula=2&snowdrop=0
```

Синтаксический анализ данных

Методы `parseJSON()` и `parseXML()` особенно полезны при обработке результатов Ajax-запросов. Для большинства веб-приложений формат JSON стал предпочтительным форматом данных по причинам, о которых говорилось в главе 14. Формат XML по-прежнему используется, но даже я стал замечать за собой, что использую XML-данные лишь в тех случаях, когда новые приложения приходится интегрировать с унаследованными серверными системами. Пример использования метода `parseJSON()` приведен в листинге 33.13.

Листинг 33.13. Преобразование данных JSON

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Сериализовать</button>")
            .appendTo("#buttonDiv").click(function(e) {

            var jsonData = '{"name": "Adam Freeman",
                "city": "London", "country": "UK"}'

            var dataObject = $.parseJSON(jsonData)

            for (var prop in dataObject) {
                console.log("Свойство: " + prop +
                    " Значение: " + dataObject[prop])
            }

            e.preventDefault();
        });
        astor=1&daffodil=0&rose=0&peony=0&primula=2&snowdrop=0
    });
</script>
...
```

В этом примере определяется простая строка в формате JSON, которая преобразуется с помощью метода `parseJSON()` в объект JavaScript. После этого просматриваются все свойства объекта, и их значения выводятся на консоль.

```
Свойство: name Значение: Adam Freeman
Свойство: city Значение: London
Свойство: country Значение: UK
```

Удаление начальных и концевых пробелов в строках

Метод `trim()` удаляет все пробельные символы в начале и конце строк. К числу пробельных символов относятся собственно пробелы, символы табуляции и разделители строк. В большинстве языков программирования такая возможность включена в основную функциональность, обеспечивающую обработку символьных данных, но по каким-то причинам отсутствует в JavaScript. Пример использования метода `trim()` приведен в листинге 33.14.

Листинг 33.14. Использование метода `trim()`

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Serialize</button>")
            .appendTo("#buttonDiv").click(function(e) {

            var sourceString =
                "\n  Эта строка содержит пробелы    ";
            console.log(">" + sourceString + "<")

            var resultString = $.trim(sourceString);
            console.log(">" + resultString + "<")

            e.preventDefault();
        });
    });
</script>
...
```

В этом примере метод `trim()` используется для удаления пробельных символов из заданной строки, и обе строки — исходная и обработанная — выводятся на консоль.

```
>
  Эта строка содержит пробелы    <
>Эта строка содержит пробелы<
```

Другие служебные методы

Существует ряд методов jQuery, которые не вписываются в какую-либо категорию, что, однако, не мешает им быть весьма полезными (табл. 33.6).

Таблица 33.6. Другие служебные методы

Метод	*Описание*
`$.contains(`*`HTMLElement1`*`,`*`HTMLElement2`*`)`	Возвращает `true`, если первый элемент содержит второй элемент
`$.proxy(`*`функция, <контекст>`*`)`	Создает функцию, контекстом которой всегда является указанный объект
`$.now()`	Возвращает текущее значение времени и представляет собой упрощенную форму вызова `newDate().getTime()`

Проверка включения элементов

Метод `contains()` позволяет проверить, содержится ли один элемент в другом. Оба его аргумента являются объектами `HTMLElement`. Если элемент, представленный первым аргументом, содержит элемент, представленный вторым аргументом, то метод возвращает значение `true`. Соответствующий демонстрационный пример приведен в листинге 33.15.

Листинг 33.15. Использование метода `contains()`

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('img').hover(function(e) {
            var elem = document.getElementById("row1");
            if ($.contains(elem, this)) {
                $(e.target).css("border", e.type ==
                    "mouseenter" ? "thick solid red" : "");
            }
        });
    });
</script>
...
```

В этом сценарии мы получаем объект с помощью программного интерфейса DOM и проверяем, содержит ли он элемент, переданный методу обработчика событий. Если это так, то элемент, с которым связано событие, заключается в рамку.

Совет. Данный метод работает лишь в отношении объектов `HTMLElement`. Если аналогичную проверку необходимо выполнить для объектов `jQuery`, можете воспользоваться методом `find()`, описанным в главе 6.

Создание функции-посредника

Метод `proxy()` позволяет создать функцию-посредник, в которой переменная `this` всегда будет указывать на объект, который вы задаете. Пример, показывающий, каким образом может быть использован метод `proxy()`, приведен в листинге 33.16.

Листинг 33.16. Использование метода `proxy()`

```
...
    <script type="text/javascript">
        $(document).ready(function() {
            $('img').hover($.proxy(handleMouse, $('img').eq(0)));

            function handleMouse(e) {
                $(this).css("border", e.type ==
                    "mouseenter" ? "thick solid red" : "");
            }
        });
    </script>
...
```

В этом примере реализуется функция `handleMouse()`, которая в ответ на события мыши устанавливает границу для элемента, представляемого переменной `this`. Я использовал метод `proxy` для помещения функции `handleMouse()` в оболочку с той целью, чтобы значение переменной `this` всегда было установлено на первый элемент `img` в документе. Эффект использования метода `proxy()` состоит в том, что, независимо от того, какой из элементов `img` является источником события мыши, граница всегда применяется к первому элементу `img`.

Совет. Этот метод не относится к числу широко используемых, и не в последнюю очередь потому, что изменяется лишь переменная `this`. В силу этого, если вы, например, привыкли определять, какой именно элемент породил событие, с помощью свойства `Event.target`, то метод `proxy()` принесет мало реальной пользы.

Резюме

В этой главе были описаны служебные методы jQuery — разнородный набор вспомогательных функций, которые либо выполняют расширенные операции над объектами jQuery, либо дополняют возможности языка JavaScript. Они относятся к той категории методов, существованию которых вы рады, когда в них возникает необходимость, но можете спокойно забыть о них в большинстве проектов веб-приложений.

ГЛАВА 34

Эффекты и CSS-фреймворк jQuery UI

В этой главе описаны два вспомогательных средства, предоставляемых библиотекой jQuery UI. Первое из них — это набор усовершенствованных методов jQuery, который не только обеспечивает анимацию цветов и видимости элементов, но и позволяет анимировать применение классов CSS. Второе средство — это CSS-фреймворк, представляющий собой набор CSS-классов, с помощью которого можно применять тему оформления jQuery UI к другим HTML-документам, тем самым придавая одинаковый внешний вид всем своим веб-приложениям. Перечень тем, рассматриваемых в данной главе, приведен в табл. 34.1.

Таблица 34.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Анимация цвета	Используйте улучшенный метод `animate()`	1
Анимация применения классов	Используйте улучшенные методы `addClass()`, `removeClass()` и `toggleClass()`, а также метод `switchClass()`	2, 3
Анимация видимости элементов	Используйте улучшенные методы `show()`, `hide()` и `toggle()`	4
Применение эффекта без изменения видимости элемента	Используйте метод `effect()`	5
Придание элементу стиля виджета	Используйте контейнерные классы `widget`	6
Применение скругленных углов к элементу	Используйте классы `corner`	7
Применение к элементу стилей виджета, способного реагировать на щелчки	Используйте классы состояния взаимодействия	8
Предоставление пользователю информации о состоянии элемента	Используйте классы `cue`	9, 10

Использование эффектов jQuery UI

Библиотека jQuery UI расширяет часть методов, входящих в состав ядра библиотеки jQuery, таким образом, чтобы обеспечивалась возможность анимации различного рода переходов между состояниями элемента. Спектр охватываемых анимацией переходов простирается от изменения цвета до применения классов

CSS. Разумно используя эти возможности, можно значительно улучшить веб-приложение, и с этой целью в jQuery UI определены некоторые дополнительные анимационные эффекты.

Анимация цвета

В библиотеке jQuery UI поддержка анимации цвета усилена за счет расширения возможностей метода jQuery `animate()`, описанного в главе 10. Можно анимировать одно из нескольких свойств CSS, определяющих цвета элемента. Список свойств, поддерживаемых методом `animate()`, приведен в табл. 34.2.

Таблица 34.2. CSS-свойства, поддерживаемые методом `animate()` jQuery UI

Свойство	*Описание*
`backgroundColor`	Устанавливает цвет фона элемента

`borderTopColor`
`borderBottomColor`
`borderLeftColor`
`borderRightColor`
	Устанавливает цвет отдельных границ элемента

`color`	Устанавливает цвет текста для элемента
`outlineColor`	Устанавливает цвет обводки; используется для визуального выделения элемента

Чтобы анимировать цвета, необходимо передать методу `animate()` в качестве аргумента объект отображения данных `map`, детализирующий свойства, которые нужно анимировать, и их целевые значения. Соответствующий пример приведен в листинге 34.1.

Листинг 34.1. Анимация цвета

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
         href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        #animTarget {
            background-color: white;
            color: black;
            border: medium solid black;
            width: 200px; height: 50px;
            text-align: center;
            font-size: 25px;
            line-height: 50px;
            display: block;
            margin-bottom: 10px;
        }
    </style>
```

```
    <script type="text/javascript">
        $(document).ready(function() {
            $('button').click(function() {
                $('#animTarget').animate({
                    backgroundColor: "black",
                    color: "white"
                })
            })
        });
    </script>
</head>
<body>
    <h1>Магазин Джеки</h1>

    <div id=animTarget>
        Привет!
    </div>

    <button>Анимировать цвет</button>
</body>
</html>
```

В этом документе для элемента `div` первоначально устанавливаются основной черный цвет и белый фон. Щелчок на кнопке приводит к вызову метода `animate()`, который изменяет эти цвета соответственно на белый и черный. Смена цветов происходит плавно, и оба цвета анимируются одновременно. Результат проиллюстрирован на рис. 34.1.

Совет. Обратите внимание на то, что в элементе `style` используются стандартные имена свойств CSS, например `background-color`. При указании того же свойства в объекте `map` мы переходим к использованию так называемого "верблюжьего регистра" (camel case) — `backgroundColor`. Это позволяет указывать CSS-свойство в качестве свойства объекта JavaScript, не заключая термин в кавычки.

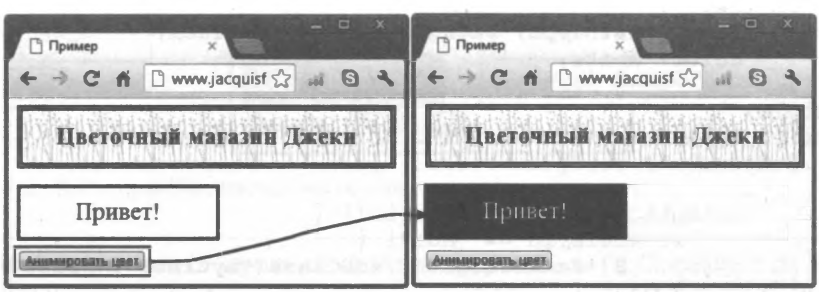


Рис. 34.1. Анимация цвета

В этом примере для указания цветов используются литералы CSS: `black` и `white`. Литеральные имена существуют для большого количества цветов, но метод `animate()` воспринимает также шестнадцатеричные значения, например `#FFFFFF`, и RGB-значения, например `rgb(255, 255, 255)`.

Совет. Несмотря на то, что в данном случае речь идет о цветовых свойствах, метод `animate()` используется точно так же, как описывалось в главе 10.

Анимация на основе классов

Библиотека jQuery UI предоставляет удобный способ анимации наборов CSS-свойств с использованием классов. Вместо того чтобы указывать каждое свойство по отдельности, вы просто определяете свойства и значения в классе и указываете jQuery UI на необходимость добавления класса в один или несколько элементов. Переход элементов из одного состояния в другое будет автоматически анимироваться библиотекой jQuery UI. Демонстрационный пример приведен в листинге 34.2.

Листинг 34.2. Анимация на основе классов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .elemClass {
            background-color: white;
            color: black;
            border: medium solid black;
            width: 200px; height: 50px;
            text-align: center;
            font-size: 25px;
            line-height: 50px;
            display: block;
            margin-bottom: 10px;
        }
        .myClass {
            font-size: 40px; background-color: black;
            color: white;
        }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('button').click(function() {
                if (this.id == "add") {
                    $('#animTarget').addClass("myClass", "fast")
                } else {
                    $('#animTarget')
                        .removeClass("myClass", "fast")
                }
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div id=animTarget class="elemClass">
```

```
        Привет!
    </div>

    <button id="add">Добавить класс</button>
    <button id="remove">Удалить класс</button>
</body>
</html>
```

Здесь мы также сталкиваемся с примером того, как jQuery UI расширяет существующий метод jQuery для наделения его дополнительной функциональностью. Стандартные версии этого метода описаны в главе 8. Версии jQuery UI делают то же самое, но теперь можно задать длительность анимации, указав ее в качестве второго аргумента при вызове метода, и jQuery UI анимирует переход из одного класса в другой.

В этом примере определен класс `myClass`, и в документе предусмотрены кнопки, добавляющие и удаляющие этот класс с использованием литерального значения `fast` для указания длительности анимации. Анимационный эффект проиллюстрирован на рис. 34.2.

Совет. Здесь действуют стандартные правила каскадирования стилей CSS, т.е. свойства, указанные в классе, применяются лишь в том случае, если он обладает наибольшей специфичностью по отношению к целевому элементу или элементам. В предыдущем примере мы установили стиль начального состояния элемента с помощью атрибута `id`, но в данном примере, для того чтобы модификации возымели эффект, для этой цели используется класс. Более подробно о каскадировании стилей CSS говорится в главе 3.

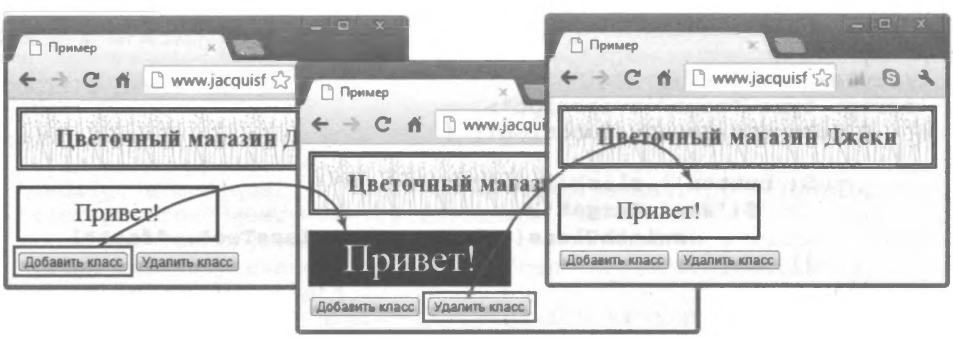


Рис. 34.2. Анимация элементов на основе классов

Совет. В библиотеке jQuery UI улучшен также метод `toggleClass()`. Он работает так же, как и стандартный метод `toggleClass()`, описанный в главе 8, но принимает дополнительный аргумент, задающий длительность анимации, и анимирует переход аналогично методам `addClass()` и `removeClass()`, работа которых была только что продемонстрирована.

Смена классов

Кроме улучшения некоторых стандартных методов, jQuery UI определяет метод `switchClass()`, который удаляет один класс и добавляет другой, одновременно анимируя переход из одного состояния в другое. Соответствующий демонстрационный пример приведен в листинге 34.3.

Листинг 34.3. Использование метода `switchClass()`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .elemClass {
            border: medium solid black;
            width: 200px; height: 50px;
            text-align: center;
            line-height: 50px;
            display: block;
            margin-bottom: 10px;
        }
        .classOne {
            font-size: 25px; background-color: white;
             color: black;
        }
        .classTwo {
            font-size: 40px; background-color: black;
            color: white;
        }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('button').click(function() {
                $('#animTarget')
                    .switchClass("classOne", "classTwo", "fast")
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div id=animTarget class="elemClass classOne">
        Привет!
    </div>
    <button>Сменить класс</button>
</body>
</html>
```

Аргументами метода `switchClass()` являются класс, подлежащий удалению, класс, который должен быть добавлен, и длительность анимации. В данном примере оба класса определяют одно и то же свойство, но это вовсе не обязательно.

Использование анимационных эффектов jQuery UI

Библиотека jQuery UI включает ряд анимационных эффектов, которые можно применять к элементам точно так же, как и эффекты ядра jQuery, описанные в главе 10. Я рекомендую применять их умеренно. Действительно, анимация, применяемая в разумных пределах, может значительность повысить комфортность работы с приложением, но нередко анимационные эффекты становятся источником раздражения и недовольства пользователей. В библиотеке имеется множество анимационных эффектов, включая `blind`, `bounce`, `clip`, `drop`, `explode`, `fade`, `fold`, `highlight`, `puff`, `pulsate`, `scale`, `shake`, `size`, and `slide`.

Примечание. В этой главе лишь показано, как применяются эти эффекты, без детального обсуждения каждого из них. Неплохое описание эффектов, а также настроек, применяемых к некоторым из них, доступно по адресу `http://docs.jquery.com/UI/Effects`.

Использование эффектов для отображения и сокрытия элементов

Библиотека jQuery UI также предоставляет улучшенные версии методов `show()`, `hide()` и `toggle()`, позволяющие создавать анимационные эффекты. Исходные версии этих методов описаны в главе 10. Чтобы использовать улучшенные версии, достаточно предоставить соответствующему методу дополнительные аргументы, задающие требуемый эффект и промежуток времени, в течение которого он должен применяться. Пример использования улучшенных версий этих методов приведен в листинге 34.4.

Листинг 34.4. Использование улучшенных методов `hide()`, `show()` и `toggle()`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet"
        type="text/css" href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .elemClass {
            font-size: 25px; background-color: white;
            color: black; border: medium solid black;
            width: 200px; height: 50px; text-align: center;
            line-height: 50px; display: block;
            margin-bottom: 10px;
        }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('button').click(function() {
                switch (this.id) {
                    case "show":
                        $('#animTarget').show("fold", "fast");
                        break;
```

```
                    case "hide":
                        $('#animTarget').hide("fold", "fast");
                        break;
                    case "toggle":
                        $('#animTarget').toggle("fold", "fast");
                        break;
                }
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <button id="hide">Скрыть</button>
    <button id="show">Показать</button>
    <button id="toggle">Переключить</button>

    <div id=animTarget class="elemClass">
        Привет!
    </div>
</body>
</html>
```

В примере предусмотрены три кнопки, щелчки на которых позволяют вызывать методы `show()`, `hide()` и `toggle()`. Для всех трех кнопок задана анимация `fold`, длительность которой определяется литеральным значением `fast`. Эти методы работают подобно их аналогам из ядра jQuery, за исключением того, что в данном случае выполняется анимация переходов.

Применение автономных эффектов

В библиотеке jQuery UI определен метод `effect()`, который позволяет применить анимацию к элементу без использования логики его "сокрытия/отображения". При удачном выборе типа анимации эта возможность является удобным способом привлечения внимания пользователя к тому или иному элементу в документе. Соответствующий демонстрационный пример приведен в листинге 34.5.

Листинг 34.5. Использование метода `effect()`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        .elemClass {
            font-size: 25px; background-color: white;
            color: black; border: medium solid black;
            width: 200px; height: 50px; text-align: center;
            line-height: 50px; display: block;
            margin-bottom: 10px;
```

```
        }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            $('button').click(function() {
                $('#animTarget').effect("pulsate", "fast")
            })
        });
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div id=animTarget class="elemClass">
        Привет!
    </div>
    <button>Эффект</button>
</body>
</html>
```

В этом примере щелчок на кнопке приводит к применению эффекта "на месте", без каких-либо постоянных изменений видимости. В данном случае используется эффект `pulsate`, вызывающий пульсацию элемента.

Использование CSS-фреймворка jQuery UI

Библиотека jQuery UI управляет внешним видом виджетов, применяя к элементам наборы классов, в которых используются сложные стили CSS. Часть классов открыта для программистов, чтобы даже тем элементам, которые не являются частью виджетов, можно было придавать стилевое оформление, одинаковое с виджетами. Некоторые из этих классов уже использовались нами в примерах в части IV.

Использование контейнерных классов виджетов

Основные стили, которые используются в виджетах, применяются с помощью трех базовых классов CSS-фреймворка. Эти классы описаны в табл. 34.3.

Таблица 34.3. Контейнерные классы виджетов jQuery UI

Класс	*Описание*
`ui-widget`	Применяется к общему внешнему контейнерному элементу всего виджета
`ui-widget-header`	Применяется к контейнерным элементам заголовков
`ui-widget-content`	Применяется к контейнерным элементам содержимого

Перечисленные классы применяются к *контейнерным* элементам, т.е. к элементам, которые целиком содержат все элементы, относящиеся к заголовку или содержимому (или, в случае класса `ui-widget`, к общему для всего виджета наружному элементу, с которым вы работаете). Применение этих классов демонстрируется в листинге 34.6.

Листинг 34.6. Использование контейнерных классов виджетов jQuery UI

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        body > div {float: left; margin: 10px}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div>
        <div>
            Цветы
        </div>
        <div>
            <div class="dcell">
                <img src="peony.png"/>
                <label for="peony">Пионы:</label>
                <input name="peony" value="0" />
            </div>
        </div>
    </div>
    <div class="ui-widget">
        <div class="ui-widget-header">
            Цветы
        </div>
        <div class="ui-widget-content">
            <div class="dcell">
                <img src="peony.png"/>
                <label for="peony">Пионы:</label>
                <input name="peony" value="0" />
            </div>
        </div>
    </div>
</body>
</html>
```

В этом примере имеются два набора элементов, к одному из которых применены контейнерные классы. Результат можно увидеть на рис. 34.3.

Скругление углов

Следующий набор классов CSS-фреймворка позволяет применять к элементам скругленные углы. Классы этой категории описаны в табл. 34.4.

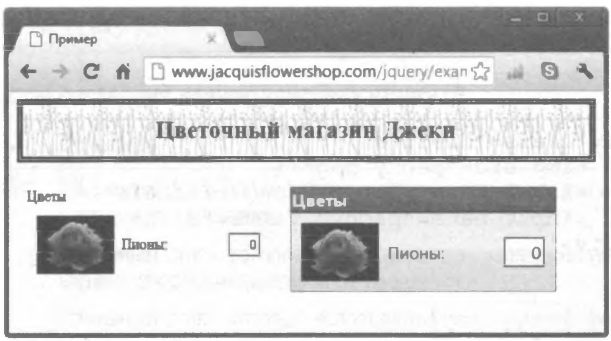


Рис. 34.3. Применение контейнерных классов виджетов jQuery UI

Таблица 34.4. Классы скругленных углов в стиле виджетов jQuery UI

Класс	*Описание*
`ui-corner-all`	Скругляет все углы элемента
`ui-corner-bl`	Скругляет левый нижний угол
`ui-corner-bottom`	Скругляет левый нижний и правый нижний углы
`ui-corner-br`	Скругляет правый нижний угол
`ui-corner-left`	Скругляет левый верхний и левый нижний углы
`ui-corner-right`	Скругляет правый верхний и правый нижний углы
`ui-corner-tl`	Скругляет левый верхний угол
`ui-corner-top`	Скругляет левый верхний и правый верхний углы
`ui-corner-tr`	Скругляет правый верхний угол

Эти классы оказывают воздействие лишь в том случае, если элемент имеет фон или поля, откуда следует, что их можно применять к классам `ui-widget-header` и `ui-widget-content`, как показано в листинге 34.7.

Листинг 34.7. Использование классов скругленных углов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        body > div {float: left; margin: 10px}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
```

```
    <div>
        <div>
            Цветы
        </div>
        <div>
            <div class="dcell">
                <img src="peony.png"/>
                <label for="peony">Пионы:</label>
                <input name="peony" value="0" />
            </div>
        </div>
    </div>

    <div class="ui-widget">
        <div class="ui-widget-header ui-corner-top"
            style="padding-left: 50px">
            Цветы
        </div>
        <div class="ui-widget-content ui-corner-bottom">
            <div class="dcell">
                <img src="peony.png"/>
                <label for="peony">Пионы:</label>
                <input name="peony" value="0" />
            </div>
        </div>
    </div>
</body>
</html>
```

Здесь суммарный эффект создается за счет скругления верхних углов контейнера заголовка (`header`) и нижних углов контейнера содержимого (`content`). Результат представлен на рис. 34.4. Обратите внимание на то, что для контейнера заголовка установлен небольшой отступ. Скругленные углы применяются внутри элемента, и поэтому во избежание отсечения содержимого для них может потребоваться дополнительное пространство.

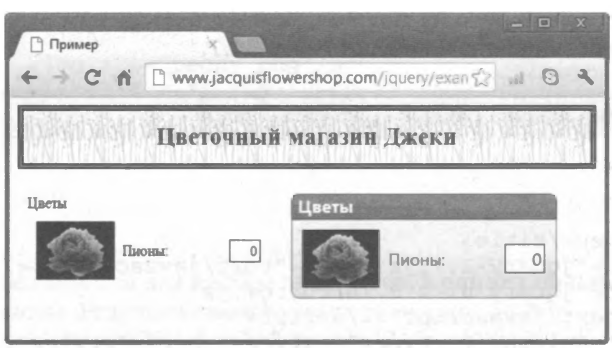


Рис. 34.4. Применение скругленных углов к элементам

Использование классов, описывающих состояние взаимодействия

Классы CSS-фреймворка можно использовать также для того, чтобы отображать различные состояния взаимодействия, что позволяет создавать элементы,

реагирующие на взаимодействие с ними пользователя так, как это делают виджеты jQuery UI. Доступные классы этой категории описаны в табл. 34.5.

Таблица 34.5. Классы состояния взаимодействия jQuery UI

Класс	*Описание*
`ui-state-default`	Применяет стиль по умолчанию, который установлен для виджетов, способных реагировать на щелчки
`ui-state-hover`	Применяет стиль, который используется при наведении указателя мыши на виджет, способный реагировать на щелчки
`ui-state-focus`	Применяет стиль, который используется при получении фокуса виджетом, способным реагировать на щелчки
`ui-state-active`	Применяет стиль, который используется для виджета, способного реагировать на щелчки, когда он активен

Пример применения перечисленных четырех классов представлен в листинге 34.8. Обратите внимание на добавление отступа для внутреннего элемента `span` во всех случаях. Классы взаимодействия определяют значения отступов, и самый простой способ создания промежутка между контейнерным элементом и содержимым — установить отступ для внутреннего элемента.

Листинг 34.8. Применение классов, описывающих состояние взаимодействия

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        body > div {float: left; margin: 10px}
        span {padding: 10px; display: block}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div class="ui-widget ui-state-default ui-corner-all">
        <span>Default</span>
    </div>
    <div class="ui-widget ui-state-hover ui-corner-all">
        <span>Hover</span>
    </div>
    <div class="ui-widget ui-state-focus ui-corner-all">
        <span>Focus</span>
    </div>
    <div class="ui-widget ui-state-active ui-corner-all">
        <span>Active</span>
    </div>
```

```
</body>
</html>
```

Результаты применения каждого класса представлены на рис. 34.5. Некоторые из состояний почти совпадают с темой оформления jQuery UI, которую использую я, но при необходимости можно воспользоваться приложением Themoroller (см. главу 17) для создания темы, которая будет визуально выделять состояние.

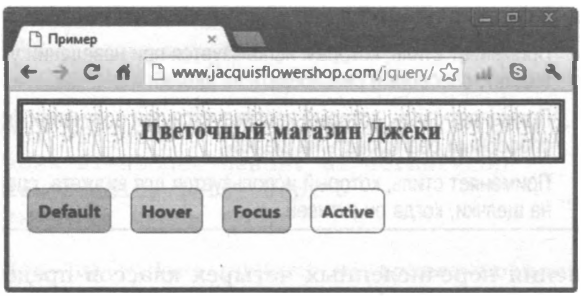


Рис. 34.5. Эффект применения классов, описывающих состояние взаимодействия

Использование классов информационных подсказок

Некоторые классы CSS-фреймворка позволяют предоставлять пользователю информационные подсказки относительно состояния элементов в документе. Эти классы описаны в табл. 34.6.

Таблица 34.6. Классы информационных подсказок jQuery UI

Класс	*Описание*
`ui-state-highlight`	Подсвечивает элемент для привлечения к нему внимания пользователя
`ui-state-error`	Визуально выделяет элемент, содержащий сообщение об ошибке
`ui-state-disabled`	Применяет стиль, соответствующий отключенной функциональности элемента (хотя в действительности функциональность элемента при этом не отключается)

Пример использования информационных подсказок `highlight` и `disabled` приведен в листинге 34.9.

Листинг 34.9. Использование информационных подсказок `highlight` и `disabled`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        body > div {float: left; margin: 10px}
```

```
        span {padding: 10px; display: block}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <div class="ui-widget">
        <div class="ui-widget-header ui-corner-top"
            style="padding-left: 5px">
            Цветы
        </div>
        <div class="ui-widget-content ui-corner-bottom">
            <div class="dcell">
                <img src="peony.png"/>
                <label for="peony">Пионы:</label>
                <input name="peony" value="0" />
            </div>
        </div>
    </div>

    <div class="ui-widget ui-state-highlight ui-corner-all">
        <div class="ui-widget-header ui-corner-top"
            style="padding-left: 5px">
            Цветы
        </div>
        <div class="ui-widget-content ui-corner-bottom">
            <div class="dcell">
                <img src="peony.png"/>
                <label for="peony">Пионы:</label>
                <input name="peony" value="0" />
            </div>
        </div>
    </div>

    <div class="ui-widget ui-state-disabled">
        <div class="ui-widget-header ui-corner-top"
            style="padding-left: 5px">
            Цветы
        </div>
        <div class="ui-widget-content ui-corner-bottom">
            <div class="dcell">
                <img src="peony.png"/>
                <label for="peony">Пионы:</label>
                <input name="peony" value="0" />
            </div>
        </div>
    </div>

</body>
</html>
```

Результаты применения этих классов проиллюстрированы на рис. 34.6. Обратите внимание на то, что одновременно с использованием класса `ui-state-highlight` я применяю стиль `ui-corner-all`. Дело в том, что указанный класс задает границу, которая по умолчанию отображается с прямыми углами. Если дочерние элементы имеют скругленные углы, то такие же углы следует задать и для выделяющего элемента.

Пример использования класса информационных подсказок `ui-state-error` приведен в листинге 34.10.

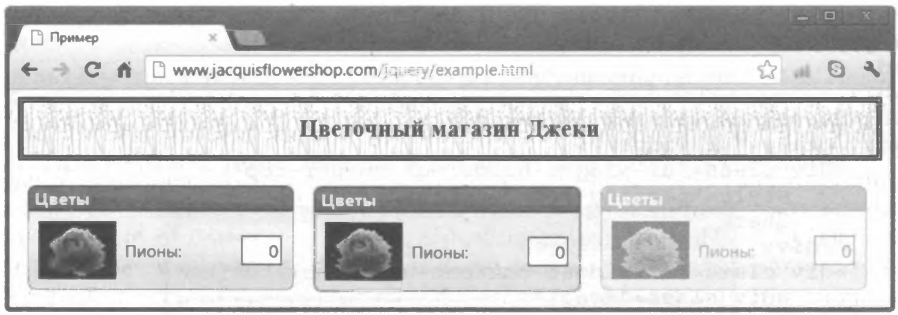


Рис. 34.6. Эффект применения классов информационных подсказок

Листинг 34.10. Использование класса `ui-state-error`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        body > div {float: left; margin: 10px; padding: 20px}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <div class="ui-state-error">
        Ой! Что-то пошло не так.
    </div>
</body>
</html>
```

Результат представлен на рис. 34.7.

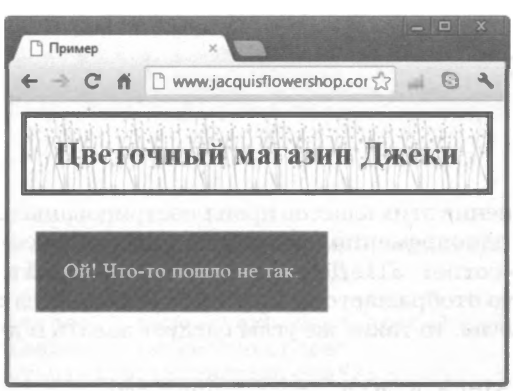


Рис. 34.7. Использование класса `ui-state-error`

Резюме

В этой главе рассматривались улучшения, предоставляемые библиотекой jQuery UI для анимации переходов, характеризующихся изменением цвета, видимости или CSS-классов. Все эти средства полезны, но их применение должно быть умеренным, чтобы на пользователя не обрушивался поток отвлекающих и раздражающих эффектов. Кроме того, были описаны наиболее важные классы CSS-фреймворка jQuery UI, обеспечивающие согласование внешнего вида элементов с внешним видом виджетов jQuery UI, благодаря чему можно распространять стилевые характеристики выбранной темы оформления jQuery UI на остальную часть ваших HTML-документов.

ГЛАВА 35

Использование отсроченных объектов

На протяжении всей книги мы постоянно сталкивались с примерами, связанными с использованием *функций обратного вызова* (callbacks). Общая схема выглядит так: вы задаете функцию, которая выполняется при возникновении определенной ситуации. Хорошим примером может служить обработка событий, когда вызывается метод наподобие `click()`, которому передается функция в качестве аргумента. Код этой функции не будет выполнен до тех пор, пока пользователь не инициирует указанное событие. До того времени функция будет неактивна.

Отсроченные объекты (deferred objects) — это термин jQuery, применяемый в отношении набора улучшений, касающихся способа использования функций обратного вызова. Отсроченные объекты позволяют применять функции обратного вызова в любых ситуациях, а не только в тех, которые связаны с событиями. При этом вам предоставляется множество опций, а также дается возможность управлять процессом выполнения собственных функций обратного вызова. Чаще всего отсроченные объекты используются для отслеживания выполнения фоновых задач, хотя это вовсе не обязательно. Данная глава начинается с относительно простого примера, который будет постепенно усложняться по мере включения в него новых функциональных возможностей. Кроме того, мы рассмотрим некоторые полезные модели управления отсроченными объектами и фоновыми задачами.

Я назвал пример, с которым мы собираемся работать, относительно простым, поскольку, приступая к использованию отсроченных объектов, вы попадаете в мир *асинхронного*, или *параллельного*, программирования. Овладение навыками параллельного программирования — задача не из легких, а язык JavaScript еще более усложняет ее, поскольку в нем отсутствуют некоторые специализированные средства, доступные в других языках, таких как Java и C#. Для большинства проектов отсроченные объекты не требуются, и если вы новичок в параллельном программировании, то рекомендую пропустить эту главу и вернуться к ней лишь тогда, когда вы начнете работать над проектом, в котором отсроченные объекты действительно будут нужны. Перечень тем, рассматриваемых в данной главе, приведен в табл. 35.1.

Таблица 35.1. Темы, рассматриваемые в данной главе

Задача	*Решение*	*Листинг*
Использование базовых возможностей отсроченного объекта	Зарегистрируйте функцию обратного вызова с помощью метода `done()`. Вызовите метод `resolve()` для запуска функции	1

Окончание табл. 35.1

Задача	Решение	Листинг
Использование отсроченного объекта с фоновой задачей	Используйте функцию `setTimeout()` для создания фоновой задачи и вызовите метод `resolve()`, когда выполнение задачи закончится	2–4
Уведомление об ошибке выполнения	Используйте метод `reject()` для запуска обработчиков, зарегистрированных с использованием метода `fail()`	5, 6
Регистрация обработчиков для перехода отсроченного объекта в любое из двух конечных состояний посредством одного вызова метода	Используйте метод `then()`	7
Определение функции, которая будет выполняться при переходе отсроченного объекта в любое из двух конечных состояний	Используйте метод `always()`	8
Использование нескольких функций обратного вызова для одного и того же перехода отсроченного объекта в любое из двух конечных состояний	Вызовите регистрирующий метод несколько раз или передайте ему список функций, разделенных запятыми	9
Создайте отсроченный объект, конечное состояние которого определяется конечными состояниями других отсроченных объектов	Используйте метод `when()`	10
Уведомление о ходе выполнения задачи	Вызовите метод `notify()`, который запустит функции обратного вызова обработчиков, зарегистрированные с помощью метода `progress()`	11, 12
Получение информации о состоянии отсроченного объекта	Используйте метод `state()`	13
Использование Ajax-объектов `Promise`	Обработайте ответ, полученный от Ajax-метода jQuery, так, словно обрабатываете отсроченный объект	14

Первый пример использования отсроченных объектов

Начнем с демонстрации того, как работают отсроченные объекты и как их можно использовать. Простой пример, содержащий отсроченный объект, приведен в листинге 35.1.

Листинг 35.1. Пример простого отсроченного объекта

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
```

```
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        td {text-align: left; padding: 5px}
        table {width: 200px; border-collapse: collapse;
            width: 50%; float: left}
        #buttonDiv {width: 15%; text-align: center;
            margin: 20px; float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            var def = $.Deferred();

            def.done(function() {
                displayMessage("Функция обратного вызова
                    выполнена");
            })

            $('button').button().click(function() {
                def.resolve();
            })

            displayMessage("Готово")
        })

        function displayMessage(msg) {
            $('tbody').append("<tr><td>" + msg + "</td></tr>")
        }
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Сообщение</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Выполнить</button>
    </div>
</body>
</html>
```

Перед вами очень простой пример, в котором используются отсроченные объекты. Мы подробно рассмотрим каждую инструкцию сценария, чтобы подготовиться к изучению остальных примеров главы. Прежде всего мы создаем отсроченный объект с помощью вызова метода `$.Deferred()`.

```
var def = $.Deferred()
```

Метод `Deferred()` возвращает отсроченный объект, который присваивается переменной `def`. Всю работу отсроченные объекты выполняют через функции обратного вызова, и поэтому наш следующий шаг — регистрация функции обратного вызова для отсроченного объекта с помощью метода `done()`.

```
def.done(function() {
    displayMessage("Функция обратного вызова
        выполнена");
})
```

В процессе своего выполнения функция обратного вызова вызывает функцию `displayMessage()`, добавляющую очередной ряд ячеек таблицы в элемент `table` документа.

Последний шаг заключается в запуске функции обратного вызова, что осуществляется посредством вызова метода `resolve()`. Этот метод переводит отсроченный объект в состояние "успешно выполнено" ("resolved"). В подобных случаях говорят о *принятии*, или *разрешении*, отсроченного объекта. Мы хотим управлять моментом перевода отсроченного объекта в состояние успешного выполнения, в связи с чем добавляем в документ кнопку и используем метод `click()` для обработки события. "Пикантность" данной ситуации состоит в том, что один механизм обратного вызова используется в качестве вспомогательного средства для описания другого. В примерах данной главы мы можем игнорировать подсистему событий и ограничиться лишь тем фактом, что отсроченный объект не будет принят до тех пор, пока не будет выполнен щелчок на кнопке. Ниже приведена функция, которая вызывает метод `resolve()` и тем самым запускает функцию обратного вызова, зарегистрированную с помощью метода `done()`.

```
$('button').button().click(function() {
    def.resolve();
})
```

До тех пор пока не вызван метод `resolve()`, отсроченный объект будет оставаться в *неразрешенном* ("unresolved") состоянии, или в состоянии "еще не выполнено" ("pending"), и функция, заданная с помощью метода `done()`, не сможет выполниться. Щелчок на кнопке приводит к смене состояния отсроченного объекта на "успешно выполнено", выполнению функции обратного вызова и отображению соответствующего сообщения в таблице, как показано на рис. 35.1.

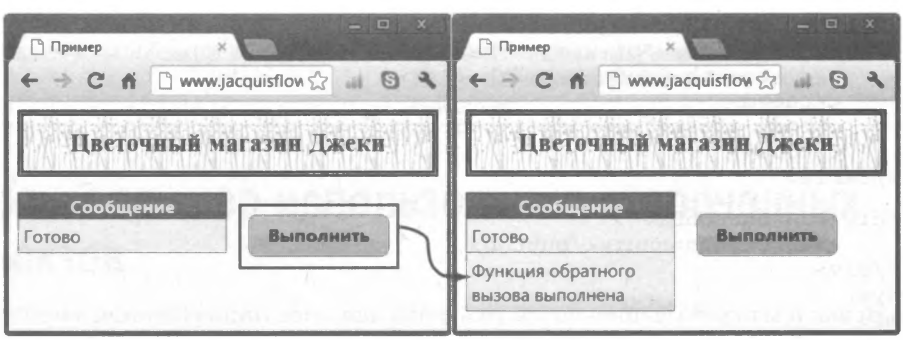


Рис. 35.1. Принятие отсроченного объекта

Здесь важно понимать, что отсроченный объект не делает ничего особенного. Вы регистрируете функцию обратного вызова с помощью метода `done()`, и она не будет выполнены до тех пор, пока не будет вызван метод `resolve()`. В этом примере отсроченный объект не будет принят до тех пор, пока на кнопке не будет выполнен щелчок, что приведет к выполнению функции обратного вызова и добавлению нового сообщения в элемент `table`.

Чем полезны отсроченные объекты

Отсроченные объекты полезны в тех случаях, когда по завершении некоторой задачи требуется выполнить определенные функции, но прямой мониторинг хода выполнения этой задачи, особенно если речь идет о фоновых задачах, нежелателен. В листинге 35.2 приведен демонстрационный пример, который мы будем постепенно видоизменять для добавления в него новых функциональных возможностей.

Листинг 35.2. Использование функций обратного вызова с длительно выполняющимися задачами

```
...
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        td {text-align: left; padding: 5px}
        table {width: 200px; border-collapse: collapse;
            width: 50%; float: left}
        #buttonDiv {width: 15%; text-align: center;
            margin: 20px; float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            var def = $.Deferred();

            def.done(function() {
                displayMessage("Функция обратного вызова
                    выполнена");
            })

            function performLongTask() {

                var start = $.now();

                var total = 0;
                for (var i = 0; i < 1075000000 ; i++) {
                    total += i;
                }
                var elapsedTime = (($.now() - start)/1000)
                    .toFixed(1)
                displayMessage("Задача завершена. Время: " +
                    elapsedTime + " с"
                def.resolve();
            }

            $('button').button().click(function() {
                displayMessage("Вызов performLongTask()")
                performLongTask()
```

```
                displayMessage("Выполнен возврат из
                    performLongTask()")
            })

            displayMessage("Готово")
        })

        function displayMessage(msg) {
            $('tbody').append("<tr><td>" + msg + "</td></tr>")
        }
    </script>

</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Сообщение</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Выполнить</button>
    </div>
</body>
</html>
...
```

В этом примере вычислительный процесс определяется функцией `performLongTask()`, выполняющей суммирование последовательности чисел. Мне хотелось выбрать для примера нечто совсем простое, чтобы на выполнение задачи уходило не более нескольких секунд, и с этой точки зрения данная задача является вполне подходящей.

Совет. В моей системе цикл `for` в функции `performLongTask()` выполняется в течение 4,7 с, но для получения аналогичного результата в своей системе вам, возможно, понадобится подставить другое значение верхнего предела счетчика цикла. Для этих примеров циклы длительностью 3–5 с представляются оптимальными. Такого времени достаточно для того, чтобы продемонстрировать возможности отсроченного объекта, и в то же время он не настолько длителен, чтобы заставить вас коротать время за чашкой кофе, дожидаясь результата.

Щелчок на кнопке приводит к вызову функции `performLongTask()`. Закончив работу, эта функция вызывает метод `resolve()` отсроченного объекта, в результате чего срабатывает функция обратного вызова. Прежде чем вызвать метод `resolve()`, функция `performLongTask()` добавляет собственное сообщение в элемент `table`, благодаря чему можно отслеживать последовательность выполнения операций в сценарии. Результат приведен на рис. 35.2.

Перед вами типичный пример *синхронной* задачи. Вы щелкаете на кнопке и ждете момента, когда каждая из вызываемых функций завершит свою работу. Пребывание кнопки Выполнить в нажатом состоянии в течение всего времени, пока функция `performLongTask()` занята вычислениями, служит наилучшим индикатором того, что задача выполняется в синхронном режиме. Убедительные доказательства синхронности вычислительного процесса предоставляет хронология

сообщений, отображенных на рис. 35.2. Сообщения от обработчика события `click` приходят как до, так и после сообщений от функции `performLongTask()` и функции обратного вызова.

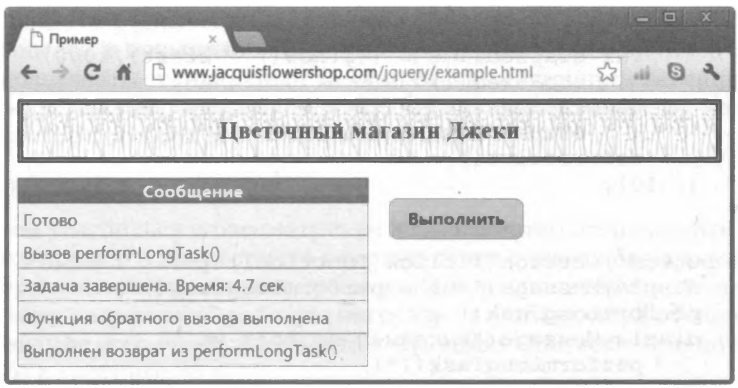


Рис. 35.2. Мониторинг хода выполнения задачи с помощью отсроченного объекта

Основные преимущества использования отсроченных объектов проявляются тогда, когда вы работаете с *асинхронными* задачами, т.е. с задачами, выполняющимися в фоновом режиме. Поскольку мы не хотим допускать никаких задержек в реакции интерфейса приложения на попытки пользователя взаимодействовать с ним, как это происходило в предыдущем примере, мы запускаем задачи в фоновом режиме, ведем мониторинг их выполнения, непрерывно следим за ними и обновляем документ, чтобы предоставить пользователю информацию о статусе выполнения задачи и результатах ее работы.

Самый простой способ запуска фоновой задачи — сделать это с помощью функции `setTimeout()`, что означает использование еще одного механизма обратного вызова. Как ни удивительно, но в JavaScript не предусмотрены средства для управления асинхронными задачами, предоставляемые в других языках программирования, и поэтому нужно обходиться теми средствами, которые имеются. В листинге 35.3 приведен пример, видоизмененный таким образом, чтобы та часть функции `performLongTask()`, на которую расходуется большая часть процессорного времени, выполнялась в фоновом режиме.

Листинг 35.3. Асинхронные вычисления

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var def = $.Deferred();

        def.done(function() {
            displayMessage("Функция обратного вызова
                выполнена");
        })

        function performLongTask() {

            setTimeout(function() {
```

```
                var start = $.now();

                var total = 0;
                for (var i = 0; i < 1075000000 ; i++) {
                    total += i;
                }
                var elapsedTime = (($.now() - start)/1000)
                    .toFixed(1)
                displayMessage("Задача завершена. Время: " +
                    elapsedTime + " с")
                def.resolve();
            }, 10);
        }

        $('button').button().click(function() {
            displayMessage("Вызов performLongTask()")
            performLongTask()
            displayMessage("Выполнен возврат из
                performLongTask()")
        })

        displayMessage("Готово")
    })

    function displayMessage(msg) {
        $('tbody').append("<tr><td>" + msg + "</td></tr>")
    }
</script>
...
```

Здесь в функцию `performLongTask()` включен вызов функции `setTimeout()`, с помощью которой перед началом выполнения цикла `for` устанавливается задержка длительностью 10 мс. Эффект внесения этого изменения представлен на рис. 35.3. Обратите внимание на то, что сообщения, поступающие от обработчика события `click`, отображаются раньше сообщений от функции `performLongTask()` и функции обратного вызова. Если вы выполните этот пример, то заметите, что кнопка возвращается в свое обычное состояние сразу же после щелчка, а не только после того, как задача завершится.

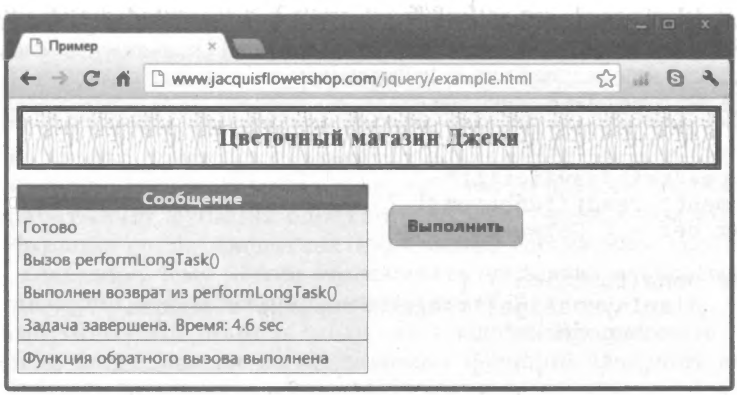


Рис. 35.3. Выполнение задачи в фоновом режиме

Значимость функций обратного вызова особенно возрастает при работе с фоновыми задачами, поскольку неизвестно, когда эти задачи завершатся. Вы могли бы настроить собственную сигнальную систему, используя, например, обновление специальной переменной, но это пришлось бы делать для каждой из выполняемых фоновых задач, что с ростом их количества привело бы к увеличению трудоемкости всего процесса и повышению вероятности ошибок. Отсроченные объекты позволяют использовать стандартный механизм для индикации завершения задач, и, как будет продемонстрировано в последующих примерах, обеспечивают необычайную гибкость в способах достижения этой цели.

Доработка примера

Прежде чем углубиться в рассмотрение возможностей отсроченных объектов, я хочу обновить пример и привести его в соответствие с тем шаблоном, которого стараюсь придерживаться в реальных проектах. Это исключительно личные предпочтения, но я хочу "вычленить" рабочую нагрузку из асинхронной оболочки и интегрировать создание отсроченного объекта в функцию. Соответствующие изменения представлены в листинге 35.4.

Листинг 35.4. Доработка примера

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        td {text-align: left; padding: 5px}
        table {width: 200px; border-collapse: collapse;
            float: left; width: 300px}
        #buttonDiv {text-align: center; margin: 20px; float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            function performLongTaskSync() {
                var start = $.now();

                var total = 0;
                for (var i = 0; i < 1075000000 ; i++) {
                    total += i;
                }
                var elapsedTime = (($.now() -
                    start)/1000).toFixed(1)
                displayMessage("Задача завершена. Время: " +
                    elapsedTime + " с")
                return total;
            }

            function performLongTask() {
                return $.Deferred(function(def) {
                    setTimeout(function() {
```

```
                    performLongTaskSync();
                    def.resolve();
                }, 10)
            })
        }

        $('button').button().click(function() {
            if ($(':checked').length > 0) {
                displayMessage("Вызов performLongTask()")
                var observer = performLongTask();
                observer.done(function() {
                    displayMessage("Функция обратного вызова
                    выполнена");
                });
                displayMessage("Выполнен возврат из
                    performLongTask()")
            } else {
                displayMessage("Вызов performLongTaskSync()")
                performLongTaskSync();
                displayMessage("Выполнен возврат из
                    performLongTaskSync()")
            }
        })

        $(':checkbox').button();
        displayMessage("Готово")
    })

    function displayMessage(msg) {
        $('tbody').append("<tr><td>" + msg + "</td></tr>")
    }
</script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Сообщение</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Выполнить</button>
        <input type="checkbox" id="async" checked>
        <label for="async">Асинхр.</label>
    </div>
</body>
</html>
```

В этом примере вся вычислительная часть выделена в функцию `performLongTaskSync()`, которая отвечает только за выполнение вычислений. О фоновых задачах или функциях обратного вызова этой функции абсолютно ничего неизвестно. Я предпочитаю выделять вычисления в отдельную функцию, поскольку это упрощает тестирование кода на ранних стадиях разработки проекта. Код синхронной функции приводится ниже.

```
function performLongTaskSync() {
    var start = $.now();

    var total = 0;
    for (var i = 0; i < 1075000000 ; i++) {
        total += i;
    }
    var elapsedTime = (($.now() -
        start)/1000).toFixed(1)
    displayMessage("Задача завершена. Время: " +
        elapsedTime + " с")
    return total;
}
```

Я также отделил код, предназначенный для асинхронного выполнения задачи. Этот код помещен в функцию `performLongTask()`, которая играет роль асинхронной оболочки для функции `performLongTaskSync()` и использует отсроченный объект для запуска функций обратного вызова по завершении вычислений. Код переработанной функции `performLongTask()` приведен ниже.

```
function performLongTask() {
    return $.Deferred(function(def) {
        setTimeout(function() {
            performLongTaskSync();
            def.resolve();
        }, 10)
    })
}
```

Если методу `Deferred()` передается некоторая функция, то она выполняет инициализацию отсроченного объекта непосредственно перед его возвращением конструктором, причем доступ к этому объекту внутри функции возможен как через первый из ее аргументов, так и через переменную `this`. Используя эту возможность, можно создать простую функцию-оболочку, которая асинхронно выполняет всю работу и после этого запускает функции обратного вызова.

Совет. Если вы были внимательны, то могли заметить, что возможны ситуации, когда вызов метода `done()` для регистрации функции обратного вызова может произойти уже после того, как задача будет выполнена и вызовется метод `resolve()`. Такое может случаться в случае очень коротких задач, но даже если метод `done()` вызывается после метода `resolve()`, функция обратного вызова все равно будет вызвана.

Другая причина, по которой я предпочитаю создавать оболочки, подобные рассмотренной выше, состоит в том, что отсроченные объекты не могут быть сброшены в исходное состояние сразу же после того, как были приняты или отклонены (вскоре я объясню, что означает *отклонение* отсроченного объекта). Создавая отсроченный объект внутри функции-оболочки, я получаю уверенность в том, что всегда использую готовые к работе, еще не выполняющиеся отсроченные объекты.

В этот пример я внес еще одно изменение, добавив кнопку, позволяющую выбирать один из двух режимов выполнения задачи: синхронный или асинхронный. В последующих примерах эта возможность отсутствует, поскольку тема данной главы — асинхронные задачи, а ее в включение в данный пример объясняется моим желанием как можно лучше объяснить читателям, в чем состоит разница между синхронным и асинхронным режимами выполнения.

Результаты работы примера в асинхронном и синхронном режимах представлены на рис. 35.4.

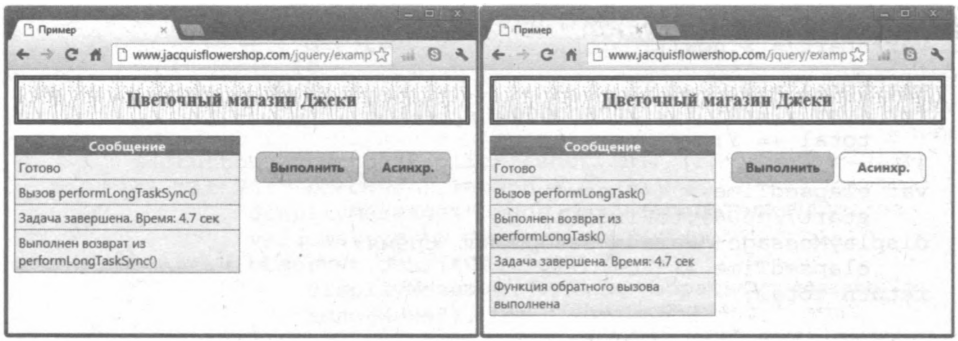


Рис. 35.4. Выполнение одной и той же задачи в синхронном и асинхронном режимах

Использование других функций обратного вызова

Теперь, когда в нашем распоряжении имеется базовый пример асинхронного выполнения задач, можем обратиться к рассмотрению других полезных возможностей, предлагаемых отсроченными объектами. Одна из них — это возможность сигнализировать о состоянии выполнения задач. В табл. 35.2 описаны методы, доступные для регистрации функций обратного вызова, а также методы, которые вызываются для запустившего их отсроченного объекта. О методах `done()` и `resolve()` уже говорилось, тогда как остальные методы рассматриваются в следующих разделах.

Таблица 35.2. Методы для регистрации функций обратного вызова

Метод	*Чем запускается*
`done()`	`resolve()`
`fail()`	`reject()`
`always()`	`resolve()` или `reject()`

Отклонение отсроченного объекта

Не все задачи выполняются успешно. В случае успешного завершения задачи вы *принимаете*, или *разрешаете*, отсроченный объект, переводя его в состояние "успешно выполнено" путем вызова метода `resolve()`. Если же что-то пошло не так, вы *отклоняете* отсроченный объект, переводя его в состояние "ошибка при выполнении" с помощью метода `reject()`[1]. Для задач, завершающихся сбоем, также можно регистрировать функции обратного вызова, для чего предназначен метод `fail()`. Функции обратного вызова, зарегистрированные с помощью метода `fail()`, запускаются методом `reject()` в полной аналогии с тем, как это делает связка методов `resolve()` и `done()`. Пример задачи, в которой отсроченный объект либо принимается, либо отклоняется, приведен в листинге 35.5.

[1] Важно подчеркнуть, что состояние отсроченного объекта может быть изменено только один раз. — *Примеч. ред.*

Листинг 35.5. Отклонение отсроченного объекта

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        td {text-align: left; padding: 5px}
        table {width: 200px; border-collapse: collapse;
            float: left; width: 300px}
        #buttonDiv {text-align: center; margin: 20px; float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            function performLongTaskSync() {
                var start = $.now();

                var total = 0;
                for (var i = 0; i < 5000000  ; i++) {
                    total += (i + Number((Math.random() +
                        1).toFixed(0)));
                }
                var elapsedTime = (($.now() -
                    start)/1000).toFixed(1)
                displayMessage("Задача завершена. Время: " +
                    elapsedTime + " с")
                return total;
            }

            function performLongTask() {
                return $.Deferred(function(def) {
                    setTimeout(function() {
                        var total = performLongTaskSync();
                        if (total % 2 == 0) {
                            def.resolve(total);
                        } else {
                            def.reject(total);
                        }
                    }, 10)})
            }

            $('button').button().click(function() {
                displayMessage("Вызов performLongTask()")
                var observer = performLongTask();
                displayMessage("Выполнен возврат из
                    performLongTask()")
                observer.done(function(total) {
                    displayMessage("Done - функция обратного
                        вызова выполнена: " + total);
                });
                observer.fail(function(total) {
```

```
                displayMessage("Fail - функция обратного
                    вызова выполнена: " + total);
            });
        })

        displayMessage("Готово")
    })

    function displayMessage(msg) {
        $('tbody').append("<tr><td>" + msg + "</td></tr>")
    }
</script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Сообщение</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Выполнить</button>
    </div>
</body>
</html>
```

В этом примере я немного подправил задачу, с тем чтобы на каждой итерации цикла `for` к общей сумме добавлялось небольшое случайное число. Асинхронная функция-оболочка `performLongTask()` проверяет сумму, возвращенную синхронной функцией, и принимает отсроченный объект, если эта сумма четная. Если же сумма оказывается нечетной, то функция `performLongTask()` отклоняет отсроченный объект.

```
...
if (total % 2 == 0) {
    def.resolve(total);
} else {
    def.reject(total);
}
...
```

После вызова функции `performLongTask()` обработчик события `click` регистрирует функции обратного вызова для обоих возможных исходов выполнения задачи (успех и неудача), используя методы `done()` и `fail()`.

```
...
var observer = performLongTask();
displayMessage("Выполнен возврат из
    performLongTask()")
observer.done(function(total) {
    displayMessage("Done - функция обратного
        вызова выполнена: " + total);
});
observer.fail(function(total) {
    displayMessage("Fail - функция обратного
        вызова выполнена: " + total);
```

```
});
...
```

Обратите внимание на то, что при вызове методов `resolve()` и `reject()` я передаю им аргументы. Поступать так вовсе не обязательно, но если вы это делаете, то предоставляемые вами объекты будут передаваться в качестве аргументов функциям обратного вызова, что позволяет обеспечивать дополнительный контекст или подробную информацию о том, что именно произошло. В данном примере состояние задачи определяется путем вычисления общей суммы, которая передается в качестве аргумента методам `done()` и `reject()`. Результаты для обоих возможных конечных состояний отсроченного объекта представлены на рис. 35.5.

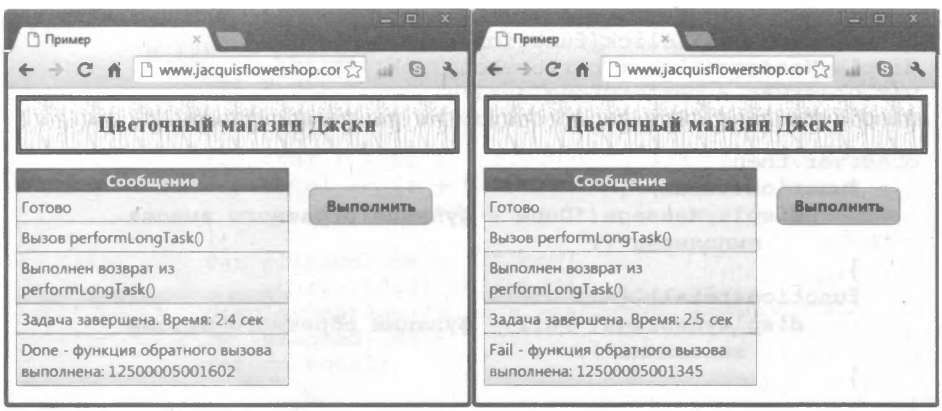


Рис. 35.5. Задача, которая может завершиться успешно или с ошибкой

Формирование цепочки вызовов для методов отсроченного объекта

Методы отсроченного объекта можно объединять в цепочки, когда каждый метод возвращает отсроченный объект, для которого вызываются другие методы. Здесь наблюдается полная аналогия с тем, как мы обращались с объектами jQuery на протяжении всей книги[2]. Пример формирования цепочки, включающей методы `done()` и `fail()`, приведен в листинге 35.6.

Листинг 35.6. Формирование цепочки вызовов для методов отсроченного объекта

```
...
$('button').button().click(function() {
    performLongTask().done(function(total) {
        displayMessage("Done - функция обратного вызова
                        выполнена: " + total);
    }).fail(function(total) {
        displayMessage("Fail - функция обратного вызова
                        выполнена: " + total);
    });
})
...
```

[2] Работая с цепочками вызовов, следует помнить о том, что объекты `jQuery` и `Deferred` имеют разные наборы методов, каждый из которых должен вызываться только для объекта своего типа. — *Примеч. ред.*

Одновременный учет обоих исходов

Если у вас предусмотрены функции обратного вызова для обоих возможных исходов выполнения задачи, то их можно зарегистрировать в одном вызове, используя метод `then()`. Первым аргументом этого метода является функция обратного вызова, которая должна использоваться для тех исходов, когда отсроченный объект принимается, а вторым — когда отсроченный объект отклоняется. Пример использования метода `then()` приведен в листинге 35.7.

Листинг 35.7. Использование метода `then()`

```
...
$('button').button().click(function() {
    displayMessage("Вызов performLongTask()")
    var observer = performLongTask();
    displayMessage("Выполнен возврат из performLongTask()")

    observer.then(
        function(total) {
            displayMessage("Done - функция обратного вызова
               выполнена");
        },
        function(total) {
            displayMessage("Fail - функция обратного вызова
                выполнена");
        }
    );
})
...
```

Обычно я использую объединение методов в цепочки, поскольку при этом получается код, в котором более четко видно, к обработке какого исхода выполнения задачи подготовлена каждая функция.

Использование функций обратного вызова, не зависящих от исхода выполнения задачи

Существуют ситуации, в которых функция обратного вызова должна выполняться независимо от того, каким был исход выполнения задачи. Обычно в этих случаях используют метод `always()`, регистрируя с его помощью функцию, которая удаляет или скрывает элементы, тем самым указывая на выполнение некоторой фоновой задачи, а также методы `done()` и `fail()`, отображающие последующие подсказки для пользователя. Пример использования метода `always()` для регистрации функции, которая ведет себя одинаковым образом при обоих исходах выполнения задачи, приведен в листинге 35.8.

Листинг 35.8. Использование метода `always()` для регистрации функции, не зависящей от исхода выполнения задачи

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
```

```
<script src="jquery-ui-1.8.16.custom.js"
    type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<link rel="stylesheet" type="text/css"
    href="jquery-ui-1.8.16.custom.css"/>
<style type="text/css">
    td {text-align: left; padding: 5px}
    table {width: 200px; border-collapse: collapse;
        float: left; width: 300px}
    #buttonDiv {text-align: center; margin: 20px; float: left}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        function performLongTaskSync() {
            var start = $.now();

            var total = 0;
            for (var i = 0; i < 5000000  ; i++) {
                total += (i + Number((Math.random() +
                    1).toFixed(0)));
            }
            var elapsedTime = (($.now() -
                start)/1000).toFixed(1)
            displayMessage("Задача завершена. Время: " +
                elapsedTime + " с")
            return total;
        }

        function performLongTask() {
            return $.Deferred(function(def) {
                setTimeout(function() {
                    var total = performLongTaskSync();
                    if (total % 2 == 0) {
                        def.resolve(total);
                    } else {
                        def.reject(total);
                    }
                }, 10)})
        }

        $('button').button().click(function() {
            displayMessage("Вызов performLongTask()")
            var observer = performLongTask();
            displayMessage("Выполнен возврат из
                performLongTask()")

            $('#dialog').dialog("open");

            observer.always(function() {
                $('#dialog').dialog("close");
            });

            observer.done(function(total) {
                displayMessage("Done -  функция обратного
                    вызова выполнена: " + total);
            });
```

```
                observer.fail(function(total) {
                    displayMessage("Fail - функция обратного
                        вызова выполнена: " + total);
                });

            })

            $('#dialog').dialog({
                autoOpen: false,
                modal: true
            })

            displayMessage("Готово")
        })

        function displayMessage(msg) {
            $('tbody').append("<tr><td>" + msg + "</td></tr>")
        }
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Сообщение</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Выполнить</button>
    </div>

    <div id="dialog">
        Выполнение задачи...
    </div>

</body>
</html>
```

В этом примере добавлено модальное окно jQuery UI, которое отображается во время выполнения задачи. Метод `always()` используется здесь для регистрации функции, которая закрывает диалоговое окно, когда задача завершена. Такой подход позволяет избавиться от дублирования кода для раздельной обработки принятых или отклоненных отсроченных объектов по завершении задачи в подобных ситуациях.

Совет. Функции обратного вызова вызываются в том порядке, в каком они зарегистрированы для отсроченного объекта. В этом примере метод `always()` вызывается до метода `done()` или `fail()`, а это означает, что функция, не зависящая от исхода, всегда вызывается до того, как будут вызваны функции, обрабатывающие исходы, в которых отсроченный объект принимается или отклоняется.

Одновременное использование нескольких функций обратного вызова

Одним из преимуществ отсроченных объектов является возможность разбиения кода на небольшие функции, обрабатывающие конкретные операции. Отсроченные объекты обеспечивают дополнительную декомпозицию кода, позволяя регистрировать несколько функций обратного вызова для одного и того же исхода выполнения задачи. Соответствующий демонстрационный пример приведен в листинге 35.9.

Листинг 35.9. Одновременная регистрация нескольких функций обратного вызова для отсроченного объекта

```
...
<script type="text/javascript">
    $(document).ready(function() {

        function performLongTaskSync() {
            var start = $.now();

            var total = 0;
            for (var i = 0; i < 5000000  ; i++) {
                total += (i + Number((Math.random() +
                    1).toFixed(0)));
            }
            var elapsedTime = (($.now() -
               start)/1000).toFixed(1)
            displayMessage("Задача завершена. Время: " +
                elapsedTime + " с")
            return total;
        }

        function performLongTask() {
            return $.Deferred(function(def) {
                setTimeout(function() {
                    var total = performLongTaskSync();
                    if (total % 2 == 0) {
                        def.resolve({
                            total: total
                        });
                    } else {
                        def.reject(total);
                    }
                    }, 10)})
            }

            $('button').button().click(function() {
                displayMessage("Вызов performLongTask()")
                var observer = performLongTask();
                displayMessage("Выполнен возврат из
                    performLongTask()")

                $('#dialog').dialog("open");
```

```
            observer.done(function(data) {
                data.touched = 1;
                displayMessage("Done 1 - функция обратного
                    вызова выполнена");
            });

            observer.done(function(data) {
                data.touched++;
                displayMessage("Done 2 - функция обратного
                    вызова выполнена");
            }, function(data) {
                data.touched++;
                displayMessage("Done 3 - функция обратного
                    вызова выполнена");
            });

            observer.done(function(data) {
                displayMessage("Done 4 - функция обратного
                    вызова выполнена: " + data.touched);
            });

            observer.fail(function(total) {
                displayMessage("Fail - функция обратного
                    вызова выполнена: " + total);
            });

            observer.always(function() {
                displayMessage("Always - функция обратного
                    вызова выполнена");
                $('#dialog').dialog("close");
            });

        })

        $('#dialog').dialog({
            autoOpen: false,
            modal: true
        })

        displayMessage("Готово")
    })

    function displayMessage(msg) {
        $('tbody').append("<tr><td>" + msg + "</td></tr>")
    }
</script>

...
```

В этом примере метод `done()` используется для регистрации четырех функций обратного вызова. Как видите, функции можно регистрировать по отдельности или группами, передавая регистрирующему методу сразу несколько функций, разделенных запятыми. Отсроченный объект гарантирует, что все функции обратного вызова будут выполнены в том порядке, в котором они были зарегистрированы.

Обратите внимание на то, что в данном примере я изменил аргумент, передаваемый методу `resolve()`, превращая результат вычислений в свойство объекта

JavaScript. Сделано это для того, чтобы продемонстрировать, как функции обратного вызова способны изменять данные, передаваемые посредством отсроченного объекта. Подобный подход может быть использован для организации связи между функциями-обработчиками (например, для оповещения о том, что были предприняты какие-то определенные действия). Результат использования групп функций-обработчиков представлен на рис. 35.6.

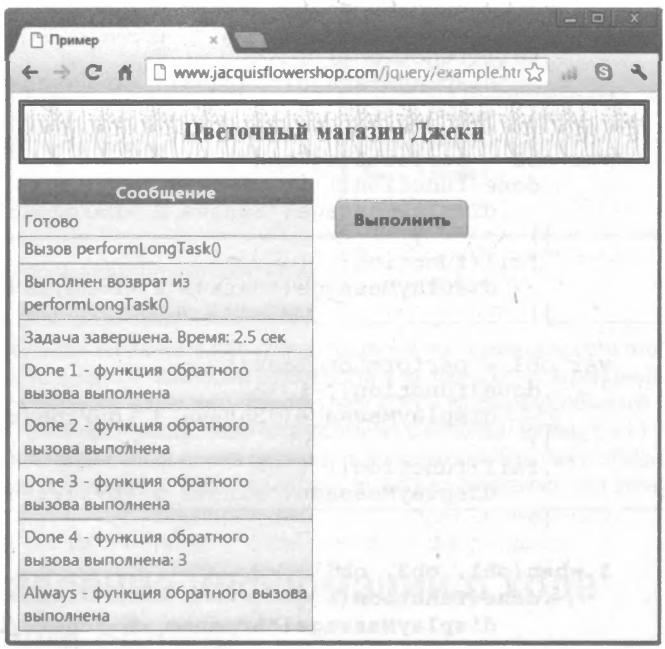


Рис. 35.6. Одновременное использование нескольких функций обратного вызова для одного и того же исхода выполнения задачи

Совет. Можно определить целую группу функций обратного вызова для каждого исхода с помощью метода `then()`, передав ему в качестве аргумента массив функций.

Проверка конечных состояний нескольких отсроченных объектов

Метод `when()` можно использовать для создания отсроченных объектов, конечное состояние которых определяется конечными состояниями нескольких других отсроченных объектов. Эта методика может быть полезной в тех ситуациях, когда решения приходится принимать на основании результатов выполнения нескольких фоновых задач или когда вы не хотите запускать задачу до тех пор, пока не будете уверены в том, что выполнение набора других задач не закончилось определенным образом. Соответствующий демонстрационный пример приведен в листинге 35.10.

Листинг 35.10. Использование метода `when()`

```
$('button').button().click(function() {

    var ob1 = performLongTask()
        .done(function() {
            displayMessage("Задача 1 <b>Успех</b>")
        })
        .fail(function() {
            displayMessage("Задача 1 <b>Неудача</b>")
        })

    var ob2 = performLongTask()
        .done(function() {
            displayMessage("Задача 2 <b>Успех</b")
        })
        .fail(function() {
            displayMessage("Задача 2 <b>Неудача</b>")
        })

    var ob3 = performLongTask()
        .done(function() {
            displayMessage("Задача 3 <b>Успех</b>")
        })
        .fail(function() {
            displayMessage("Задача 3 <b>Неудача</b>")
        })

    $.when(ob1, ob2, ob3)
        .done(function() {
            displayMessage("Агрегат <b>Успех</b>")
        })
        .fail(function() {
            displayMessage("Агрегат <b>Неудача</b>")
        })
})
...
```

В этом примере имеются три отсроченных объекта, каждый из которых был создан посредством вызова функции `performLongTask()` и с которыми я связал функции обратного вызова с помощью методов `done()` и `fail()`.

Все три отсроченных объекта передаются методу `when()`, который возвращает еще один отсроченный объект (так называемый *агрегатный отсроченный объект*). Для связывания функций обратного вызова с агрегатным отсроченным объектом используются обычные методы `done()` и `fail()`. Конечное состояние агрегатного объекта определяется совокупностью конечных состояний трех других отсроченных объектов. Если *все три* обычных отсроченных объекта приняты, то принимается и агрегатный объект, и в этом случае будут вызываться функции обратного вызова, зарегистрированные методом `done()`. Но если *хотя бы один* из обычных отсроченных объектов отклонен, то отклоняется и агрегатный объект, вследствие чего будут вызываться функции обратного вызова, зарегистрированные методом `fail()`. Результаты для обоих исходов представлены на рис. 35.7.

Рис. 35.7. Использование метода `when()`

Предупреждение. Если вы внимательно присмотритесь к последовательности отображения сообщений на рисунке, то заметите некоторые особенности в их хронологии. Агрегатный отсроченный объект отклоняется сразу же после того, как отклонен любой из базовых объектов. Это означает, что запуск функций обратного вызова, зарегистрированных с помощью метода `fail()`, может осуществляться даже в то время, когда другие задачи еще выполняются. Работая с отклоненным агрегатным объектом, нельзя предполагать, будто все задачи, от которых он зависит, уже завершены.

Предоставление информации о ходе выполнения задачи

Обычно в случае длительно выполняющихся фоновых задач никогда не бывает лишним предоставлять пользователю информацию о ходе выполнения задачи. Отсроченные объекты можно использовать для передачи этой информации из задачи функциям обратного вызова во многом так же, как и при передаче данных об исходе выполнения задачи. Сведения о ходе выполнения получают с помощью метода `notify()`, а для регистрации функций обратного вызова используют метод `progress()`. Соответствующий пример приведен в листинге 35.11.

Листинг 35.11. Предоставление и использование информации о ходе выполнения задачи посредством отсроченного объекта

```
...
<script type="text/javascript">
    $(document).ready(function() {

        function performLongTaskSync() {
            var total = 0;
            for (var i = 0; i < 5000000 ; i++) {
                total += (i + Number((Math.random() +
                    1).toFixed(0)));
            }
```

```
            return total;
        }

        function performLongTask() {
            return $.Deferred(function(def) {
                setTimeout(function() {
                    var progressValue = 0;
                    for (var i = 0; i < 4; i++) {
                        performLongTaskSync();
                        progressValue += 25;
                        def.notify(progressValue)
                    }
                    def.resolve();
                }, 10)}
            )
        }

        $('button').button().click(function() {

            performLongTask().progress(function(val) {
                displayMessage("Прогресс: " + val + "%")
            }).done(function() {
                displayMessage("Задача выполнена");
            })
        })

        displayMessage("Готово")
    })

    function displayMessage(msg) {
        $('tbody').append("<tr><td>" + msg + "</td></tr>")
    }
</script>
...
```

В этом примере вычисления, предусмотренные задачей, выполняются четыре раза. После каждого вычисления я вызываю метод `notify()` для отсроченного объекта и сообщаю ему процент выполнения задачи (можно передавать любой другой объект или значение, соответствующие специфике конкретного веб-приложения). В обработчике события `click` метод `progress()` используется для регистрации функции, которая будет вызываться в ответ на обновление индикатора процесса. Эта функция используется для добавления сообщения в элемент `table` документа.

В данном примере демонстрируются базовые возможности предоставления информации о ходе выполнения, но не все в нем работает так, как хотелось бы. Проблема состоит в том, что информация об изменениях, которую браузер должен использовать для обновления DOM новыми строками таблицы, будет получена им только после того, как завершатся все четыре итерации. Это является следствием того, как в JavaScript осуществляется управление задачами. Отсюда следует, что обновления информации о ходе выполнения задачи будут получены вами в виде одной порции данных, когда закончится выполнение всех итераций. Для устранения этого недостатка мы организуем небольшую задержку на каждой из стадий выполнения задачи, чтобы браузер имел возможность оперативно обновлять информацию. Пример использования для этих целей функции `setTimeout()` и цепочки отсроченных объектов приведен в листинге 35.12. Для настройки задержек и отсроченных объектов я мог бы использовать цикл `for`, но, чтобы сделать пример нагляднее, я определил все четыре стадии в явном виде.

Листинг 35.12. Разбиение задачи на отдельные этапы для отображения изменений в DOM

```
...
<script type="text/javascript">
    $(document).ready(function() {

        function performLongTaskSync() {
            var total = 0;
            for (var i = 0; i < 5000000  ; i++) {
                total += (i + Number((Math.random() +
                    1).toFixed(0)));
            }
            return total;
        }

        function performLongTask() {

            function doSingleIteration() {
                return $.Deferred(function(innerDef) {
                        setTimeout(function() {
                        performLongTaskSync();
                        innerDef.resolve();
                    }, 10)
                })
            }

            var def = $.Deferred();

            setTimeout(function() {
                 doSingleIteration().done(function() {
                    def.notify(25);
                    doSingleIteration().done(function() {
                        def.notify(50);
                        doSingleIteration().done(function() {
                            def.notify(75);
                            doSingleIteration().done(function() {
                                def.notify(100);
                                def.resolve();
                            })
                        })
                    })
                })
            }, 10);
            return def;
        }
         $('button').button().click(function() {
             performLongTask().progress(function(val) {
                displayMessage("Progress: " + val + "%")
            }).done(function() {
                displayMessage("Задача выполнена");
            })
        })
         displayMessage("Ready")
    })

    function displayMessage(msg) {
        $('tbody').append("<tr><td>" + msg + "</td></tr>")
    }
```

```
</script>
...
```

После описанных изменений информация о ходе выполнения задачи отображается именно так, как нам хотелось бы. Результат представлен на рис. 35.8.

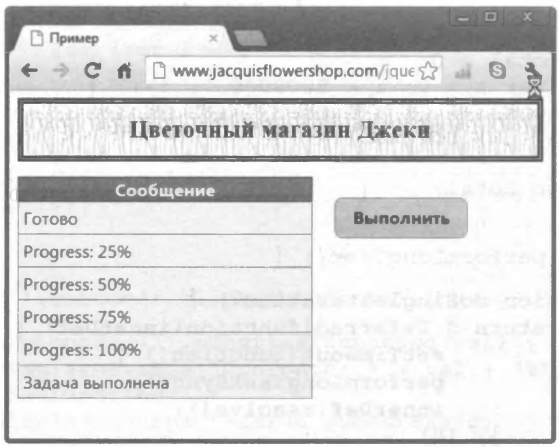


Рис. 35.8. Использование отсроченного объекта для предоставления информации о ходе выполнения

Получение информации об отсроченном объекте

Для отсроченных объектов определен метод `state()`, который можно использовать для определения состояния объекта, а значит, и выполняющейся задачи. Возвращаемые этим методом значения перечислены в табл. 35.3.

Таблица 35.3. Значения, описывающие состояние объекта

Значение	*Описание*
`pending`	Ни один из методов `resolve()` и `reject` не вызывался для данного объекта
`resolved`	Отсроченный объект был принят (с помощью метода `resolve()`)
`rejected`	Отсроченный объект был отклонен (с помощью метода `reject()`)

Предупреждение. Применяйте этот метод осмотрительно. Планируя опрашивать состояние отсроченного объекта, задумайтесь, все ли в вашем проекте веб-приложения для этого учтено. Добавление операций, связанных с опросом состояния, особенно в циклах `while` и `for`, может сделать задачу фактически синхронной, хотя при этом с нею будут по-прежнему связаны дополнительные накладные расходов и сложности, присущие асинхронным задачам.

С моей точки зрения, метод `state()` может быть полезным лишь в одном случае, когда нас интересует исход выполнения задачи, а функция обратного вызова зарегистрирована с помощью метода `always()`. Обычно я регистрирую отдельные наборы функций обратного вызова с помощью методов `done()` и `fail()`, но иногда код для обоих исходов оказывается практически одним и тем же с незначительными отличиями. Пример использования метода `state()` приведен в листинге 35.13.

Листинг 35.13. Использование метода `state()`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        td {text-align: left; padding: 5px}
        table {width: 200px; border-collapse: collapse;
            float: left; width: 300px}
        #buttonDiv {text-align: center; margin: 20px; float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            function performLongTaskSync() {
                var start = $.now();

                var total = 0;
                for (var i = 0; i < 5000000  ; i++) {
                    total += (i + Number((Math.random() +
                        1).toFixed(0)));
                }
                var elapsedTime = (($.now() -
                    start)/1000).toFixed(1)
                displayMessage("Задача завершена. Время: " +
                    elapsedTime + " с")
                return total;
            }

            function performLongTask() {
                return $.Deferred(function(def) {
                    setTimeout(function() {
                        var total = performLongTaskSync();
                        if (total % 2 == 0) {
                            def.resolve(total);
                        } else {
                            def.reject(total);
                        }
                    }, 10)})
            }

            $('button').button().click(function() {
                displayMessage("Вызов performLongTask()")
                var observer = performLongTask();
                displayMessage("Выполнен возврат из
                    performLongTask()")

                $('#dialog').dialog("open");

                observer.always(function() {
                    if (observer.state() == "resolved") {
```

```
                    $('#dialog').dialog("close");
                } else {
                    $('#dialog').text("Ошибка!")
                }
            });

            observer.done(function(total) {
                displayMessage("Done - функция обратного
                    вызова выполнена: " + total);
            });
            observer.fail(function(total) {
                displayMessage("Fail - функция обратного
                    вызова выполнена: " + total);
            });

        })

        $('#dialog').dialog({
            autoOpen: false,
            modal: true

        })

        displayMessage("Готово")
    })

    function displayMessage(msg) {
        $('tbody').append("<tr><td>" + msg + "</td></tr>")
    }
</script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Сообщение</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Выполнить</button>
    </div>

    <div id="dialog">
        Выполнение задачи...
    </div>
</body>
</html>
```

Использование отсроченных объектов Ajax

Возможно, наиболее полезным аспектом функциональности отсроченных объектов является то, что они включены в предусмотренные в jQuery средства поддержки Ajax (см. главы 14 и 15). Объект `jqXHR`, возвращаемый такими методами,

как ajax() и getJSON(), реализует интерфейс Promise, который предоставляет подмножество методов, определяемых обычным отсроченным объектом. Интерфейс Promise определяет методы done(), fail(), then() и always() и может использоваться вместе с методом when(). Пример совместного использования объектов Promise и обычных отсроченных объектов приведен в листинге 35.14.

Совет. Вам предоставляется возможность создавать собственные объекты Promise, вызывая метод promise() для отсроченного объекта. Это может пригодиться, если вы пишете библиотеку JavaScript и хотите, чтобы другие программисты могли лишь связывать с объектами свои функции обратного вызова, но не принимать или отклонять ваши отсроченные объекты.

Листинг 35.14. Использование объектов Promise и обычных отсроченных объектов

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-1.7.js" type="text/javascript"></script>
    <script src="jquery-ui-1.8.16.custom.js"
        type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css"
        href="jquery-ui-1.8.16.custom.css"/>
    <style type="text/css">
        td {text-align: left; padding: 5px}
        table {width: 200px; border-collapse: collapse;
            float: left; width: 300px}
        #buttonDiv {text-align: center; margin: 20px; float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            function performLongTaskSync() {
                var start = $.now();

                var total = 0;
                for (var i = 0; i < 5000000  ; i++) {
                    total += (i + Number((Math.random() +
                        1).toFixed(0)));
                }
                var elapsedTime = (($.now() -
                start)/1000).toFixed(1)
                displayMessage("Задача завершена. Время: " +
                    elapsedTime + " с")
                return total;
            }

            function performLongTask() {
                return $.Deferred(function(def) {
                    setTimeout(function() {
                        performLongTaskSync();
                        def.resolve();
                    }, 10)})
            }
```

```
            $('button').button().click(function() {
                displayMessage("Вызов performLongTask()")
                var observer = performLongTask().done(function() {
                    displayMessage("Задача завершена")
                });
                displayMessage("Выполнен возврат из
                    performLongTask()")

                displayMessage("Вызов getJSON()")
                var ajaxPromise = $.getJSON("mydata.json")
                    .done(function() {
                    displayMessage("Ajax-запрос выполнен")
                });
                displayMessage("Выполнен возврат из getJSON()")

                $.when(observer, ajaxPromise).done(function() {
                    displayMessage("Все задачи выполнены");
                })
            })
            displayMessage("Готово")
        })

        function displayMessage(msg) {
            $('tbody').append("<tr><td>" + msg + "</td></tr>")
        }
    </script>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Сообщение</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Выполнить</button>
    </div>
</body>
</html>
```

В этом примере используется метод `getJSON()`, а результат обрабатывается так, как если бы это был отсроченный объект. Функция обратного вызова подключается с помощью метода `done()` и используется в качестве аргумента при вызове метода `when()`. Результаты работы этого примера показаны на рис. 35.9.

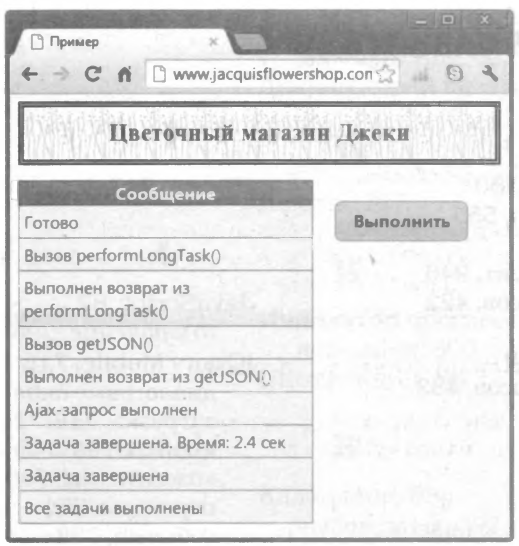


Рис. 35.9. Использование объектов интерфейса `Promise`

Резюме

В этой главе были продемонстрированы возможности отсроченных объектов jQuery, с помощью которых можно сигнализировать о ходе выполнения, а также об успешном или неудачном завершении задач, как правило, тех, которые выполняются в фоновом режиме. Отсроченные объекты используются также в рамках предоставляемой в jQuery поддержки Ajax, которая обеспечивает унифицированный способ обработки Ajax-запросов и фоновых пользовательских задач. Отсроченные объекты обеспечивают дополнительные функциональные возможности, необходимости в которых в большинстве веб-приложений не возникает, но в проектах, связанных с интенсивным использованием фоновых задач, они способны значительно повысить интерактивность приложений.

Предметный указатель

О

П

С

Т

У

Ф

Ц

Ш

Э