Кит Вуд

Расширение библиотеки jQuery

Keith Wood

Extending jQuery

Foreword by Dave Methvin

MANNING
SHELTER ISLAND

Кит Вуд

Расширение библиотеки jQuery

С предисловием Дэйва Метвина

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2014

УДК 004.738.5:004.45jQuery
ББК 32.973.202-018.2
В88

Кит Вуд
В88 Расширение библиотеки jQuery / пер. с анг. Киселева А.Н. – М.: ДМК Пресс, 2014. – 400 с.: ил.

ISBN 978-5-97060-071-9

jQuery – одна из наиболее популярных библиотек для разработки клиентских сценариев на JavaScript. В ней предусмотрено большое количество точек интеграции, посредством которых можно добавлять собственные селекторы и фильтры, расширения, анимационные эффекты и многое другое. Эта книга покажет вам, как это делается.

Из книги вы узнаете, как писать расширения и как проектировать их, чтобы максимально обеспечить возможность их многократного использования. Вы также научитесь писать новые виджеты и эффекты для jQuery UI. Наряду с этим вы исследуете особенности создания расширений для применения в таких ключевых аспектах библиотеки, как технология Ajax, события, анимация и проверка данных.

Издание предназначено для веб-программистов разной квалификации, уже использующих jQuery в своей работе.

УДК 004.738.5:004.45jQuery
ББК 32.973.202-018.2

ISBN 978-1-617291-03-6 (анг.)
ISBN 978-5-97060-071-9 (рус.)

Содержание

Предисловие

С момента появления в 2006 году jQuery превратилась в самую популярную библиотеку JavaScript, используемую для управления документами HTML и наполнения их динамическими особенностями. Врожденная поддержка самых разных браузеров в jQuery позволяет разработчикам сосредоточиться на создании веб-сайтов, а не на реализации обходных решений, учитывающих характерные особенности браузеров. В 2013 году более половины из миллиона самых посещаемых веб-сайтов использовали jQuery. Аналогично библиотека jQuery UI, основанная на библиотеке jQuery, стала самым популярным источником виджетов пользовательского интерфейса.

С ростом популярности у разработчиков jQuery стало появляться искушение расширить библиотеку, чтобы с ее помощью можно было решать практически любые проблемы, с которыми приходится сталкиваться программистам. Однако добавление новых возможностей в ядро jQuery влечет за собой увеличение объема кода, загружаемого посетителями веб-сайтов, независимо от того, используются эти возможности на данном сайте или нет. Создание такой огромной, монолитной библиотеки неизбежно привело бы к ухудшению производительности в угоду удобству веб-разработчиков, что является далеко не самым лучшим компромиссом.

Чтобы предотвратить «разбухание» исходного кода, разработчиками jQuery была выработана философия, согласно которой в библиотеку помещается только наиболее востребованная функциональность и предоставляется возможность ее расширения. Удивительная экосистема расширений для jQuery продолжала расти все эти годы, движимая потребностью разработчиков внести свой вклад и стремлением поделиться своими наработками с обширным сообществом пользователей jQuery. Успех jQuery в значительной степени обусловлен этими благородными порывами, и команда проекта jQuery всячески способствует этому, предоставляя площадки для размещения кода, такие как сайт plugins.jquery.com.

Кит Вуд (Keith Wood) прекрасно подходит на роль наставника по расширению библиотеки jQuery. Он является завсегдатаем форума jQuery Forum (forum.jquery.com), где пользуется заслуженным авторитетом благодаря своим исчерпывающим ответам и решениям проблем, с которыми сталкиваются разработчики. Кроме того, его авторитет подкрепляется разработанными им популярными расшире-

ниями для jQuery. То есть Кит не только обладает обширными практическими знаниями в области расширения jQuery, но и понимает, какие особенности jQuery заслуживают подробного описания, а не мимолетного упоминания.

Эта книга детально описывает практически все аспекты расширения библиотеки jQuery, независимо от того, создаются эти расширения для личного использования или для коммерческого распространения. Наиболее простыми являются расширения методов библиотеки jQuery Core, но в книге также пристальное внимание уделяется расширениям для библиотеки виджетов jQuery UI, которые часто создаются для улучшения визуального представления информации. Детальность документации с описанием фабрики виджетов в библиотеке jQuery UI часто оказывается недостаточной, что еще больше увеличивает ценность соответствующих глав.

Мне особенно приятно отметить, что Кит уделил достаточное внимание модульным тестам. Кто-то может посчитать создание исчерпывающего комплекта модульных тестов напрасной тратой времени, но такое мнение обычно держится до момента, когда спустя несколько месяцев безобидное, казалось бы, изменение в расширении вынуждает всю команду тратить долгие часы на отладку на действующем сайте, под непрерывным потоком жалоб от пользователей. Модульные тесты не способны выявить все возможные ошибки, но они позволяют убедиться в работоспособности и предотвратить наиболее очевидные проблемы, которые легко могут пропустить нетерпеливые разработчики при тестировании вручную.

Независимо от причин, которые подтолкнули вас к изучению темы расширения возможностей jQuery, подумайте о том, чтобы поделиться своими наработками с сообществом, если вам покажется, что они смогут принести пользу кому-нибудь. Это полностью соответствует философии jQuery. Делясь своими знаниями, вы не только помогаете другим, но и зарабатываете профессиональный авторитет.

Дэйв Метвин (Dave Methvin),
президент jQuery Foundation

Вступление

Впервые я столкнулся с библиотекой jQuery в начале 2007 года и сразу же понял, насколько она проста и удобна в использовании. С ее помощью мне быстро удалось реализовать поиск элементов, их сокрытие и отображение. Затем я попробовал задействовать некоторые сторонние расширения и обнаружил, что они сильно отличаются своей практической ценностью и удобством использования.

Мне посчастливилось включиться в разработку расширения, ставшего впоследствии одним из самых популярных в сообществе jQuery. Я наткнулся на виджет Clean Calendar Марка Грабански (Marc Grabanski) на JavaScript, который он преобразовал в расширение для jQuery, и мне понравился его интерфейс ввода даты. Я начал экспериментировать с ним, добавлять новые особенности и попутно исследовать возможности jQuery. Результаты своих экспериментов я предложил Марку включить в его расширение. Так началась наша совместная работа над расширением, продолжавшаяся в течение следующих нескольких лет.

К тому моменту расширение Calendar было переименовано в Datepicker и выбрано командой проекта jQuery UI в качестве основы для виджета выбора даты.

Начиная с того момента я занялся разработкой других расширений, в которых испытывал потребность или к которым у меня возникал интерес. В числе моих расширений, завоевавших популярность, можно назвать: альтернативное расширение Datepicker, позволяющее выбирать диапазон дат или несколько отдельных дат; расширение Calendars с поддержкой календарей, отличных от григорианского; расширение Countdown, отображающее время, оставшееся до указанной даты и времени; и расширение SVG Integration, позволяющее взаимодействовать с SVG-элементами на веб-страницах. В процессе создания этих расширений я хорошо изучил язык JavaScript и библиотеку jQuery, а также тонкости создания расширений для последней.

Разработка расширений – идеальный способ зафиксировать функциональные возможности в форме программных компонентов многократного пользования и обеспечить простоту их внедрения в другие веб-страницы. Это подталкивает вас гораздо тщательнее тестировать свой код и гарантировать непротиворечивость его поведения в любых окружениях.

За прошедшие годы библиотека jQuery значительно выросла в размерах и в количестве функциональных возможностей, но она оста-

лась верной своей цели – максимально упростить труд разработчика. Разрастающееся сообщество разработчиков расширений – яркое свидетельство, что расширяемая архитектура для библиотеки jQuery была выбрана правильно. Я надеюсь, что взгляд изнутри на разработку расширений, представленный в этой книге, позволит вам максимально использовать возможности jQuery в своих проектах.

Благодарности

В первую очередь я хочу выразить благодарность Джону Резигу (John Resig) и команде проекта jQuery за создание такого ценного инструмента для разработчиков.

Спасибо также Марку Грабански (Marc Grabanski) за предоставленную возможность участвовать в разработке расширения Calendar/Datepicker, с которого началась моя карьера разработчика расширений.

Книга – всегда плод усилий многих людей, и мне хотелось бы выразить благодарность коллективу редакторов издательства Manning: Берту Бейтсу (Bert Bates), Фрэнку Полманну (Frank Pohlmann) и Синтии Кейн (Cynthia Kane); корректорам: Рензо Холлхамеру (Renso Hollhumer) Михиль Тримпи (Michiel Trimpe); и всем остальным, участвовавшим в подготовке книги к печати, за поддержку и наставление. Отдельное спасибо Кристине Рудлофф (Christina Rudloff) за начальное предложение от лица издательства Manning написать книгу о библиотеке jQuery UI.

Спасибо всем разработчикам, присылавшим мне свои комментарии, предложения, отчеты об ошибках и файлы локализации моих расширений. Особое спасибо всем, кто помогал мне в моей работе, – я обожаю музыку и танцы!

Я очень благодарен рецензентам первых вариантов моей рукописи за отзывы, которые помогли улучшить конечный результат: Амандип Джасвал (Amandeep Jaswal), Энни Эпштейн (Anne Epstein), Брэди Келли (Brady Kelly), Бруно Фигуеиредо (Bruno Figueiredo), Дэниел Миди (Daniele Midi), Дэвид Уокер (David Walker), Есиль Теодоро (Ecil Teodoro), Джерейнт Уильямс (Geraint Williams), Джузеппе де Марко (Giuseppe De Marco), доктор Йорг Иезекиль Бо (PhD, Jorge Ezequiel Bo), Лиза Морган (Lisa Z. Morgan), Майк Ма (Mike Ma), Пим ван Хьювен (Pim Van Heuven) и Стивен Райс (Stephen Rice).

Отдельное спасибо Дэйву Метвину (Dave Methvin), президенту jQuery Foundation, за предисловие и одобрение моей книги.

И напоследок, но не в последнюю очередь искренне благодарю мою супругу Тресили (Trecialee) за терпение в течение всего времени работы над этим проектом (даже при том, что она не разбирается в теме).

Об этой книге

jQuery – самая широко используемая в Сети библиотека JavaScript, предлагающая массу возможностей и существенно облегчающая разработку веб-приложений. Но она концентрируется на предоставлении только наиболее востребованных и часто используемых функциональных возможностей и не в состоянии охватить все, что может вам потребоваться. Вы можете сами встраивать дополнительные функции в свои веб-страницы, но если вдруг обнаружится, что один и тот же код повторяется от страницы к странице, возможно, пора подумать о создании расширения для jQuery.

Расширение позволяет заключить программный код в единственный модуль, который затем можно будет применять к любому количеству веб-страниц. Преимущество единой библиотеки расширений заключается в уменьшении затрат на тестирование и сопровождение, а также в единстве внешнего вида и поведения всех страниц вашего сайта.

Библиотека jQuery изначально проектировалась с поддержкой расширений, чтобы дать им возможность стать полноправными членами окружения jQuery и использоваться наряду со встроенной функциональностью. Данная книга расскажет вам, как написать расширение, легко интегрирующееся с библиотекой, не порождающее конфликтов с другими расширениями и предоставляющее гибкое и надежное решение некоторой задачи.

Кому адресована эта книга?

Эта книга рассказывает о расширении библиотеки jQuery и о создании модулей расширений (плагинов). Читателем может быть технический специалист, желающий узнать, что можно расширить в библиотеке jQuery, чтобы наладить создание модулей многократного пользования для своих проектов. Или им может быть веб-разработчик, стремящийся освоить тонкости разработки надежного кода для jQuery. Или сторонний разработчик расширений, желающий создавать плагины для передачи сообществу jQuery.

Учитывая особенности целевой аудитории, предполагается, что вы имеете некоторое знакомство с jQuery. Вы, как ожидается, умеете с помощью jQuery выбирать элементы и выполнять операции над их свойствами, отображать или скрывать элементы и подключать к ним обработчики событий. Вы должны иметь опыт применения сущест-

вующих сторонних расширений для добавления функциональности в свои страницы.

Не имеющим опыта работы с библиотекой jQuery я рекомендую предварительно прочитать книгу «jQuery in Action, Second Edition» Бера Бибо (Bear Bibeault) и Иегуды Каца (Yehuda Katz) (Manning, 2010)[1].

jQuery – это библиотека на JavaScript, поэтому также предполагается, что вы знакомы с этим языком программирования. Основную часть расширений составляет код на JavaScript, содержащий вызовы методов из библиотеки jQuery или обращения к точкам интеграции. В программном коде часто используются такие конструкции, как анонимные функции, тернарные операторы и даже замыкания. Поэтому будет здорово, если эти термины знакомы вам. В противном случае, возможно, вам стоит сначала освежить свои знания языка JavaScript.

Подробное описание языка JavaScript можно найти в книге Джона Резига (John Resig) и Бера Бибо (Bear Bibeault) «Secrets of the JavaScript Ninja» (Manning 2012).

Краткое содержание

Книга «Расширение библиотеки jQuery» делится на четыре части. Часть I (главы 1–3) охватывает простые расширения, добавляющие в библиотеку jQuery новые возможности. Часть II (главы 4–7) описывает наиболее удачные подходы к реализации расширений и функций. Часть III (главы 8–10) рассматривает вопросы расширения библиотеки jQuery UI, используемой для создания пользовательского интерфейса. Часть IV (главы 11–14) охватывает множество других немаловажных тем: анимационные эффекты, применение технологии Ajax, обработка событий и расширение Validation, которое не является частью jQuery, но играет важную роль.

- Глава 1 коротко рассказывает об истории jQuery и обсуждает возможные направления расширения библиотеки.
- Глава 2 охватывает модули, составляющие jQuery, и более подробно рассказывает о возможностях их расширения. Затем демонстрирует основные приемы разработки расширений на примере создания простого плагина.
- Глава 3 показывает, как расширять селекторы jQuery, с помощью которых выполняется поиск элементов веб-страниц.

---

[1] Бер Бибо, Иегуда Кац. jQuery. Подробное руководство по продвинутому JavaScript. – 2-е изд. – ISBN: 978-5-93286-201-8. – Символ-Плюс, 2011.

- Глава 4 отступает на шаг назад и рассказывает о передовых приемах разработки, помогающих создавать надежные и полезные расширения.
- Глава 5 демонстрирует создание расширения коллекций на основе принципов, описанных в предыдущей главе. Расширения коллекций воздействуют сразу на множество элементов страницы.
- Глава 6 рассматривает функциональные расширения, реализующие дополнительные возможности, не связанные с какими-то конкретными элементами, например локализация приложения или обработка блоков cookie.
- Глава 7 обсуждает приемы тестирования и упаковки, гарантирующие корректную работу расширений и упрощающие их получение и использование. В ней также описывается, как документировать и демонстрировать расширение, чтобы пользователи могли применять все его возможности.
- Глава 8 показывает, как на основе библиотеки jQuery UI создавать расширения коллекций, позволяющие объединять компоненты jQuery UI в единый виджет.
- Глава 9 объясняет, как с помощью модуля jQuery UI Mouse реализовать рисование с помощью мыши внутри собственного расширения.
- Глава 10 завершает часть книги, посвященную библиотеке jQuery UI, знакомством с возможностями создания визуальных эффектов и корректировки частоты изменения анимируемых свойств.
- Глава 11 показывает, как передавать значения для анимируемых свойств, не являющиеся простыми числовыми значениями, на примере CSS-свойства `background-position`.
- Глава 12 рассказывает о возможностях поддержки технологии Ajax в библиотеке jQuery и показывает, как расширить их с помощью своих фильтров, транспортов и преобразователей.
- Глава 13 обсуждает особенности фреймворка обработки событий в jQuery и как его использовать для создания новых и расширения существующих событий jQuery.
- Глава 14 показывает, как расширить плагин Validation и добавить в него дополнительные правила проверки, которые могли бы применяться к отдельным элементам наряду со встроенными.

Соглашения об оформлении программного кода и комплект загружаемых примеров

В этой книге приводится множество листингов программного кода на JavaScript, а также разметки HTML и правил CSS. Исходный код в листингах и в тексте будет оформляться моноширинным шрифтом, чтобы его можно было отличить от обычного текста. Имена переменных и функций в тексте также будут оформляться подобным образом. **Жирным моноширинным шрифтом** будут выделяться ключевые фрагменты кода, обычно имена функций или переменных. Некоторые листинги подверглись переформатированию с целью уместить их по ширине книжной страницы. Большинство листингов включают дополнительные аннотации для выделения наиболее важных частей. Во многих случаях нумерованным указателям в коде соответствуют дополнительные в следующем ниже тексте.

jQuery и jQuery UI – это открытые библиотеки, распространяемые на условиях лицензии MIT[1] и могут быть загружены на веб-сайтах http://jquery.com/ и http://jqueryui.com/ соответственно. Исходный код примеров для этой книги доступен для загрузки на веб-сайте издательства Manning: http://www.manning.com/ExtendingjQuery.

Автор в Сети

Одновременно с покупкой книги «Расширение библиотеки jQuery» вы получаете бесплатный доступ к частному веб-форуму, организованному издательством Manning Publications, где можно оставлять комментарии о книге, задавать технические вопросы, а также получать помощь от автора и других пользователей. Чтобы получить доступ к форуму и зарегистрироваться на нем, откройте в веб-браузере страницу http://www.manning.com/ExtendingjQuery. Здесь описывается, как попасть на форум после регистрации, какие виды помощи доступны и правила поведения на форуме.

Издательство Manning обязуется предоставить своим читателям место встречи, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Но со стороны автора отсутствуют какие-либо обязательства уделять форуму какое-то определенное внимание – его присутствие на форуме оста-

---

[1] Текст лицензии можно найти по адресу: https://github.com/jquery/jquery/blob/master/MIT-LICENSE.txt. (См. также http://ru.wikipedia.org/wiki/Лицензия_MIT. – *Прим. перев.*)

ется добровольным (и неоплачиваемым). Мы предлагаем задавать автору стимулирующие вопросы, чтобы его интерес не угасал!

Об авторе

Кит Вуд (Keith Wood) – разработчик с 30-летним стажем, использующий библиотеку jQuery с 2007 года. Им было написано более 20 расширений – включая оригинальный плагин Datepicker, World Calendar and Datepicker, Countdown и SVG, – которые он передал сообществу пользователей jQuery. Он часто отвечает на вопросы, касающиеся библиотеки jQuery, на форумах и в 2012 году вошел в пятерку самых активных участников.

На своей основной работе он занимается созданием веб-приложений с использованием Java/J2EE на стороне сервера и jQuery на стороне клиента. Проживает в Австралии, в городе Сидней, со своей супругой Тресили (Trecialee). В свободное от работы время увлекается танцами.

Об иллюстрации на обложке

На первой странице обложки книги «Расширение библиотеки jQuery» изображена «доленцианка» – жительница городка Доленци (Dolenci), расположенного на границе Словении и Венгрии. Эта иллюстрация взята из последнего издания книги Балтазара Аке (Balthasar Hacquet) «Images and Descriptions of Southwestern and Eastern Wenda, Illyrians, and Slavs», опубликованного этнографическим музеем в городе Сплит, Хорватия, в 2008 году. Балтазар Аке (1739–1815) – австрийский врач и ученый, который провел много лет, изучая ботанику, геологию и этнографию в разных частях Австрийской империи, а также в области Венето (Veneto), Италия, Юлийский Альпах и западных Балканах, населяемых в прошлом людьми самых разных племен и национальностей. Его научные труды сопровождаются многочисленными иллюстрациями, нарисованными им собственноручно.

Богатое разнообразие рисунков в трудах Аке служит ярким свидетельством уникальности и индивидуальности Альпийского и Балканского регионов всего 200 лет тому назад. Это было время, когда по одежде можно было отличить двух людей, проживающих в разных регионах, расположенных на расстоянии нескольких миль, и когда по одежде легко было определить, к какому племени или социальному классу принадлежал человек. С тех пор мода сильно изменилась, и исчезло столь богатое разнообразие. Сейчас зачастую сложно отличить жителей разных континентов, а жители итальянских Альп мало чем отличаются от жителей остальной части Европы.

Мы в издательстве Manning славим изобретательность, предприимчивость и радость компьютерного бизнеса обложками книг, изображающими богатство региональных различий двухвековой давности.

Часть I

Простые расширения

Будучи наиболее широко используемой библиотекой JavaScript, jQuery предлагает массу функций, упрощающих разработку пользовательских интерфейсов. Но вы можете сделать библиотеку jQuery еще лучше, расширяя ее дополнительными возможностями многократного пользования.

Глава 1 коротко рассказывает об истории развития jQuery, а затем показывает, какие ее элементы можно расширить. Она завершается несколькими примерами имеющихся расширений, демонстрирующими широту возможностей.

В главе 2 вы найдете описание архитектуры jQuery и доступных точек интеграции, с подробным описанием каждой из них. Затем мы познакомимся с процессом разработки простого расширения, которое можно будет немедленно включить в работу.

Простейшими расширениями, которые вам может понадобиться создавать, являются селекторы – строительные блоки, на которых основан механизм поиска элементов для выполнения последующих операций над ними. О селекторах рассказывается в главе 3, где также приводятся многочисленные примеры создания собственных селекторов.

Глава 1

Расширения для jQuery

Эта глава охватывает следующие темы:

- происхождение и назначение jQuery;
- что в библиотеке jQuery доступно для расширения;
- примеры существующих расширений.

В настоящее время jQuery является самой распространенной библиотекой JavaScript в Сети. Она предлагает множество функциональных возможностей, упрощающих разработку пользовательских интерфейсов, таких как обход дерева объектной модели документа HTML (Document Object Model, DOM) в поисках элементов для последующей обработки и применения анимационных эффектов к ним. Кроме того, разработчики jQuery понимают, что невозможно (да и не нужно) предусмотреть все необходимое, и предусмотрели точки интеграции, посредством которых можно добавлять в библиотеку новые функциональные возможности. Такая дальновидность способствовала росту популярности jQuery.

В этой книге я расскажу, как расширять различные аспекты jQuery, чтобы упростить возможность повторного использования расширений и их сопровождение. Помимо стандартных расширений, воздействующих на коллекции элементов веб-страниц, есть возможность создавать собственные селекторы, вспомогательные функции, анимационные эффекты, дополнительные механизмы обработки для использования в технологии Ajax, собственные события и правила проверки данных. Я покажу, как выполнять тестирование, упаковку и документирование кода, чтобы другие разработчики смогли максимально использовать его возможности.

1.1. История развития jQuery

На веб-сайте проекта библиотека jQuery характеризуется как «быстрая, маленькая и богатая возможностями библиотека JavaScript. Она позволяет выполнять обход элементов документа HTML, производя

операции над ними, обрабатывать события, воспроизводить анимационные эффекты и существенно упрощает использование технологии Ajax, предоставляя простой в использовании API, одинаковый для самых разных браузеров» (http://jquery.com).

Это библиотека функций JavaScript, обеспечивающих простой доступ к HTML DOM, его исследование и изменение, дающих возможность делать веб-страницы более динамичными и выразительными в соответствии с парадигмой Web 2.0. Главными особенностями библиотеки являются:

- выбор элементов с использованием CSS-подобного синтаксиса;
- возможность обхода элементов;
- поддержка операций с элементами, включая удаление и изменение содержимого и атрибутов;
- обработка событий, в том числе нестандартных;
- визуальные и анимационные эффекты;
- поддержка Ajax;
- возможность расширения функциональности (что является темой этой книги);
- различные вспомогательные функции;
- поддержка браузеров разных типов и сокрытие несовместимостей между ними.

jQuery является открытой и свободно распространяемой библиотекой. В настоящее время она распространяется на условиях лицензии MIT (http://jquery.org/license/[1]). Прежние ее версии распространялись также на условиях лицензии GNU General Public License, Version 2[2].

1.1.1. Происхождение

Первая версия jQuery была создана Джоном Резигом (John Resig) и анонсирована в январе 2006 года на конференции BarCamp NYC[3]. Он случайно наткнулся на библиотеку Behaviour, написанную Беном Ноланом (Ben Nolan), и оценил потенциал заложенных в нее идей – использование CSS-подобных селекторов для привязки различных

---

[1] http://ru.wikipedia.org/wiki/Лицензия_MIT. – *Прим. перев.*

[2] Перевод текста лицензии GNU GPL v2 можно найти по адресу: http://jxself.org/translations/gpl-2.ru.shtml. – *Прим. перев.*

[3] Джон Резиг (John Resig), «BarCampNYC Wrap-up», http://ejohn.org/blog/barcampnyc-wrap-up/.

функций JavaScript к элементам DOM. Но Джону не понравилась многословность синтаксиса и отсутствие иерархических селекторов[1]. Предложенный им синтаксис и последующая реализация легли в основу библиотеки jQuery.

В листинге 1.1 демонстрируется код, использующий библиотеку Behaviour для подключения обработчика события щелчка мышью ко всем элементам `li`, находящимся в элементе с атрибутом `id="example"`; обработчик события удаляет элемент, на котором выполнен щелчок. В листинге 1.2 демонстрируется аналогичный код, использующий библиотеку jQuery.

Листинг 1.1. Пример использования библиотеки Behaviour

```
Behaviour.register({
    '#example li': function(e){
        e.onclick = function(){
            this.parentNode.removeChild(this);
        }
    }
});
```

Листинг 1.2. Эквивалентный код, использующий библиотеку jQuery

```
$('#example li').bind('click', function(){
    $(this).remove();
});
```

Как было выбрано имя jQuery? Первоначально библиотеке было дано имя jSelect, отражающее возможность выбирать элементы вебстраницы. Но когда Джон выполнил поиск по этому имени, он обнаружил, что оно уже занято, и присвоил библиотеке имя jQuery[2].

1.1.2. Развитие

С момента первоначального анонса было выпущено множество версий библиотеки jQuery, которые перечислены в табл. 1.1 (здесь представлены не все версии). С годами она выросла весьма существенно как с точки зрения функциональности, так и с точки зрения размеров.

Несмотря на существенный рост размеров библиотеки jQuery, при минификации кода (удалении ненужных комментариев и пробельных символов) ее размер уменьшается в три раза (после минифика-

---

[1] Джон Резиг (John Resig), «Selectors in Javascript», http://ejohn.org/blog/selectors-in-javascript/.

[2] Комментарии Джона Резига, «BarCampNYC Wrap-up», http://ejohn.org/blog/barcampnyc-wrap-up/.

Таблица 1.1. Версии библиотеки jQuery (не все)

Версия	Дата выпуска	Размер, Кбайт	Примечания
1.0	26.08.2006	44,3	Первая стабильная версия
1.0.4	12.12.2006	52,2	Последняя исправленная версия 1.0
1.1	14.01.2007	55,6	Улучшение производительности селекторов
1.1.4	23.08.2007	65,6	jQuery могла быть переименована
1.2	10.09.2007	77,4	
1.2.6	26.05.2008	97,8	
1.3	13.01.2009	114	Встроен новый механизм селекторов Sizzle, добавлено динамическое управление событиями, перестроен механизм управления событиями
1.3.2	19.02.2009	117	
1.4	13.01.2010	154	Улучшение производительности, расширение поддержки технологии Ajax
1.4.1	25.01.2010	156	Добавлены методы `height()`, `width()` и `parseJSON()`
1.4.2	13.02.2010	160	Улучшение производительности, добавлен метод `delegate()`
1.4.3	14.10.2010	176	Переписан модуль CSS, добавлена обработка метаданных
1.4.4	11.11.2010	178	
1.5	31.01.2011	207	Управление отложенными обратными вызовами, переписан модуль Ajax, улучшена производительность механизма обхода дерева DOM
1.5.2	31.03.2011	214	
1.6	02.05.2011	227	Существенно увеличена производительность функций `attr()` и `val()`, добавлена функция `prop()`
1.6.4	12.09.2011	232	
1.7	03.11.2011	243	Новые функции в Event API: `on()` и `off()`, улучшена производительность механизма делегирования событий
1.7.2	21.03.2012	246	
1.8.0	09.08.2012	253	Переписан механизм селекторов Sizzle, переработан механизм поддержки анимации, улучшена модульность библиотеки
1.8.3	13.11.2012	261	
1.9.0	14.01.2013	261	Чистка кода, подготовка к выпуску версии jQuery 2.0
1.9.1	04.02.2013	262	Исправление ошибок и регрессий
2.0.0	18.04.2013	234	Исключена поддержка IE 6–8
1.10.0	24.05.2013	267	Синхронизация возможностей с веткой 2.x
1.10.2	03.07.2013	266	
2.0.3	03.07.2013	236	

ции размер последней версии уменьшается до 91 Кбайта). Если минифицированная версия возвращается веб-сервером в формате gzip, ее размер уменьшается еще примерно в три раза – ориентировочно до 32 Кбайт в последней версии. При использовании сети доставки содержимого (Content Delivery Network, CDN) этот файл может сохраняться в кэше клиента, избавляя от необходимости повторно загружать его.

Использование CDN

Чтобы загрузить библиотеку jQuery с одного из серверов CDN, включите один из тегов `script`, представленных в этой врезке. Вам может потребоваться изменить номер загружаемой версии, выбрав наиболее подходящую для ваших требований.

Использование сети CDN, поддерживаемой компанией MediaTemple

```
<script src="http://code.jquery.com/jquery-1.9.1.min.js">
</script>
```

Вы можете также включить загрузку расширения jQuery Migration с этого же сайта, который поможет выполнить переход с более старых версий jQuery на версию jQuery 1.9 или выше.

```
<script
    src="http://code.jquery.com/jquery-migrate-1.1.1.min.js">
</script>
```

Использование сети CDN, поддерживаемой компанией Google[1]

```
<script
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.
min.js">
</script>
```

В сети CDN компании Google доступны все версии jQuery, но эта сеть неподконтрольна разработчикам jQuery, и более свежие версии могут появляться в ней с некоторым опозданием.

Использование сети CDN, поддерживаемой компанией Microsoft[2]

```
<script
    src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.9.1.min.js">
</script>
```

В сети CDN компании Microsoft CDN доступны все версии jQuery, но эта сеть неподконтрольна разработчикам jQuery, и более свежие версии могут появляться в ней с некоторым опозданием.

---

[1] Список библиотек в сети CDN компании – руководство разработчика» https://developers.google.com/speed/libraries/devguide#jquery.

[2] ASP.NET, «Сеть доставки содержимого компании Microsoft» http://www.asp.net/ajaxlibrary/cdn.ashx.

В настоящее время jQuery включает механизм селекторов *Sizzle*, реализующий фундаментальную возможность поиска элементов DOM. Всякий раз, когда это возможно, механизм Sizzle делегирует обработку селекторов браузеру, но при необходимости возвращается к собственной реализации на JavaScript, чтобы обеспечить равноценную поддержку во всех основных браузерах.

1.1.3. Современное состояние дел

jQuery стала самой популярной библиотекой JavaScript в Интернете и используется на значительном количестве веб-сайтов. Она официально поддерживается компанией Microsoft и распространяется в составе комплекта продуктов Visual Studio. Компания BuiltWith сообщает, что более 60% из 10 000 самых посещаемых веб-сайтов используют jQuery, и более 50% – из миллиона[1]. Компания W3Techs сообщает, что jQuery используется на 55% всех веб-сайтов и на 90% сайтов, где вообще используется хоть какая-нибудь библиотека JavaScript[2].

Сообщество разработчиков расширений постоянно ширится, и многие из них открывают свободный доступ к своему коду в духе самой библиотеки jQuery. Вы можете самостоятельно поискать необходимые вам модули или воспользоваться постоянно обновляемым «официальным» репозиторием расширений для jQuery (http://plugins.jquery.com). Некоторые расширения просто великолепны – содержат надежный код, сопровождаются исчерпывающей документацией и примерами. Другие не так хороши, сложны в использовании, содержат ошибки и/или плохо документированы. Когда вы прочитаете эту книгу и будете следовать описываемым в ней принципам, ваши расширения, безусловно, попадут в первую категорию.

Активная жизнь также кипит на форумах, посвященных jQuery (https://forum.jquery.com), где можно найти более 250 000 ответов на более чем 110 000 вопросов. На форумах вы найдете специальные разделы, где обсуждаются вопросы создания и использования расширений.

Разработка библиотеки jQuery в настоящее время продолжается под эгидой организации jQuery Foundation (http://jquery.org). Она

---

[1] Статистика использования jQuery от компании BuiltWith: http://trends.builtwith.com/javascript/jQuery.

[2] Статистика использования библиотек JavaScript на сайте компании W3Techs: http://w3techs.com/technologies/overview/javascript_library/all.

была сформирована в сентябре 2009 года с целью поддержки всех проектов линейки jQuery, включая jQuery Core, jQuery UI, jQuery Mobile, Sizzle и QUnit. Финансовую основу этой поддержки составляют взносы и пожертвования сообщества jQuery.

1.2. Расширение jQuery

Если библиотека jQuery обладает столь широкими функциональными возможностями, зачем еще больше расширять их? Чтобы удержать объем кода jQuery в разумных пределах, в ядро библиотеки включены только наиболее востребованные и универсальные функции (хотя до сих пор не утихают дебаты о том, что считать наиболее востребованным). Базовые функции доступа к элементам, обработка событий, анимация и поддержка Ajax – вот что требуется большинству пользователей, при этом сохраняется возможность добавлять более специализированные функции.

К счастью, команда проекта jQuery прекрасно понимает, что ядро jQuery не может вместить все, что только может потребоваться, поэтому было создано множество точек интеграции для расширения функциональности jQuery и использования существующей инфраструктуры и возможностей.

Упаковка реализованного расширения для jQuery в виде плагина позволит вам повторно использовать новые функциональные возможности на других веб-страницах. В результате вам придется поддерживать единственную копию расширения, и любые последующие улучшения в нем будут немедленно вступать в силу везде, где оно используется. Вы можете организовать тестирование своего плагина в изолированном окружении и при определенных условиях, гарантирующих нормальную работу расширения.

1.2.1. Что доступно для расширения?

Несмотря на значительное разнообразие функциональных возможностей, предоставляемых ядром библиотеки, всегда можно найти массу аспектов, функциональность которых можно было бы расширить. Те аспекты, которые охватываются в этой книге, перечислены в следующих разделах.

Селекторы и фильтры

Селекторы и фильтры jQuery позволяют идентифицировать и отбирать элементы веб-страницы для выполнения последующих опера-

ций с ними. Несмотря на наличие стандартных селекторов, отбирающих узлы по именам, атрибутам `id` и `class`, иногда бывает желательно иметь дополнительные селекторы псевдоклассов (расширяющих набор псевдоклассов CSS), чтобы можно было фильтровать прежде выбранное множество элементов кратким и непротиворечивым способом. Можно также добавлять фильтры, которые принимают коллекцию предварительно отобранных элементов и фильтруют ее, опираясь на позиции отдельных элементов в этой коллекции. Создание селекторов данных типов описывается в главе 3.

Создавая собственные селекторы, можно реализовать процедуру выбора в одном месте и с легкостью использовать везде, где угодно, гарантируя единообразие реализации во всех проектах. Такой подход также упрощает сопровождение селектора и немедленное применение любых исправлений или расширений ко всем веб-страницам, где он используется.

Расширения коллекций

Расширения коллекций – это функции, которые могут применяться к коллекциям элементов, полученным с помощью селекторов. Именно эти функции подразумеваются многими под термином *расширение для jQuery*, и именно они образуют самую многочисленную группу расширений. Возможности расширений коллекций ограничиваются только вашей фантазией. Им подвластно все – от простого изменения атрибутов элементов до изменения поведенческих характеристик за счет обработки событий в этих элементах и полной замены оригинальных компонентов альтернативной реализацией.

В главе 4 представлено несколько правил, которыми желательно руководствоваться при создании расширений, а в главе 5 описывается инфраструктура поддержки расширений, которую я использую для создания собственных расширений, и демонстрируется ее соответствие упомянутым выше правилам. Правила аккумулируют в себе опыт, накопленный при создании расширений, они помогают легко интегрировать собственные расширения с библиотекой jQuery и уменьшают вероятность конфликтов с внешним кодом.

Ключевым пунктом в процессе разработки собственных расширений является их тестирование, и применение инструментов модульного тестирования позволит вам без особых усилий тестировать свой код и убеждаться в его работоспособности. Как только код будет готов к выпуску, его необходимо упаковать в дистрибутив, чтобы другие разработчики могли легко получить его и интегрировать в свои

проекты. Вы должны также создать веб-страницу, демонстрирующую возможности расширения, чтобы позволить предполагаемым пользователям познакомиться с его возможностями и особенностями. И не забудьте про документацию, описывающую все аспекты применения вашего расширения, которая даст возможность другим максимально использовать его потенциал. Все эти стороны разработки расширений освещаются в главе 7.

Расширения-функции

Расширения-функции – это вспомогательные функции, которые не выполняют операции непосредственно с коллекциями элементов. Они реализуют дополнительные возможности и обычно используют механизмы библиотеки jQuery для выполнения своей работы. В главе 6 подробно рассказывается, как создавать такие функции.

В качестве примеров подобных расширений можно привести вывод отладочных сообщений в консоль или извлечение и изменение значений cookie для веб-страницы. Реализуя такие возможности в виде расширений jQuery, вы предоставляете пользователям знакомый способ вызова кода и уменьшаете вероятность конфликтов с внешним кодом. Некоторые правила, упоминавшиеся выше, применимы и к расширениям этого вида, как и рекомендации, касающиеся тестирования, упаковки, демонстрации и документирования.

Виджеты jQuery UI

jQuery UI – «это организованный набор инструментов взаимодействий с пользовательским интерфейсом, эффектов, виджетов и тем оформления, опирающихся на основную библиотеку jQuery» (http://jqueryui.com/). Библиотека jQuery UI определяет инфраструктуру, позволяющую создавать виджеты, действующие непротиворечивым образом и способные использовать огромное количество тем оформления пользовательского интерфейса. В главе 8 рассказывается об инфраструктуре поддержки виджетов и особенностях ее использования для создания собственных визуальных компонентов.

Инфраструктура виджетов jQuery UI также следует правилам, описываемым в главе 4, и предоставляет функциональность, общую для всех виджетов jQuery UI. Принимая эту инфраструктуру в качестве основы для своих расширений, вы автоматически получаете весь ее потенциал и возможность сконцентрироваться на создании функциональности, уникальной для вашего виджета. Если вы задействуете в своем виджете классы, определенные в ThemeRoller, он ав-

томатически будет следовать общему стилю оформления с другими компонентами jQuery UI и изменяться при применении новой темы.

Некоторые виджеты jQuery UI предусматривают возможность взаимодействий с указателем мыши, и команда проекта jQuery UI понимает всю важность таких взаимодействий. Наследуя в своем виджете модуль jQuery UI Mouse вместо базового модуля Widget, вы автоматически получаете поддержку операций с мышью, позволяющую настраивать все аспекты взаимодействий, и вы снова можете сконцентрироваться на реализации особенностей, уникальных для вашего виджета. Глава 9 описывает, как задействовать модуль Mouse при создании виджета, функционирование которого основано на использовании мыши.

Эффекты jQuery UI

В состав jQuery UI входит также множество эффектов, которые могут применяться к элементам страницы. Эффекты, такие как `blind`, `clip`, `fold` и `slide`, можно использовать для отображения или сокрытия элементов. Некоторые эффекты, такие как `highlight` и `pulsate`, предназначены для привлечения внимания к элементу. Вы можете определить собственные эффекты и применять их к элементам наряду со стандартными. Глава 10 демонстрирует порядок создания новых эффектов.

Анимация свойств

В jQuery имеется механизм *анимации*, который можно задействовать для изменения любых атрибутов элементов, принимающих обычные числовые значения. Он позволяет организовать поэтапное изменение атрибута от одного значения до другого, управлять продолжительностью изменения и величиной одного шага. Но если значение, которое требуется изменять, не является простым числовым значением, вам придется реализовать необходимую функциональность самостоятельно. Например, в составе jQuery UI имеется модуль, дающий возможность изменять цвет от одного значения до другого. В главе 11 мы реализуем анимацию для атрибута, имеющего составное значение.

Поддержка AJAX

Поддержка технологии *Ajax* в библиотеке jQuery является одним из очевидных ее преимуществ. Она позволяет легко загружать удаленные данные и обрабатывать их. Обращаясь к механизму Ajax,

вы можете указать тип данных, которые готов принять ваш обработчик успешного завершения вызова: простой текст, HTML, XML, JSON. Преобразование потока байтов, поступающих от удаленной стороны, будет выполнено библиотекой автоматически. Вы можете добавлять собственные преобразования для поддержки специализированных форматов простым указанием типа данных, который должен быть возвращен. Глава 12 обсуждает порядок расширения механизма поддержки Ajax для обработки распространенных форматов файлов.

Обработка событий

Средства обработки событий в библиотеке jQuery дают возможность подключать к элементам множество обработчиков для обработки действий пользователя, системных и собственных событий. В jQuery имеется несколько точек входа для определения нестандартных событий и их возбуждения, благодаря чему обеспечивается тесная интеграция с существующей функциональностью. Глава 13 описывает реализацию нового события с целью упрощения взаимодействий с мышью.

Правила проверки

Расширение Validation, написанное Йерном Зафферером (Jörn Zaefferer), широко используется для проверки ввода пользователя на стороне клиента перед отправкой значений на сервер. Несмотря на то что это расширение не входит в ядро библиотеки jQuery, оно также предоставляет точки для подключения дополнительных расширений, благодаря которым можно определять собственные правила проверки и включать их в существующую процедуру обработки. Глава 14 иллюстрирует, как создавать собственные правила проверки и передавать их встроенному механизму для выполнения.

1.3. Примеры расширений

В Интернете можно найти сотни расширений для библиотеки jQuery, добавляющих новые функциональные возможности в веб-страницы. Их количество – самое веское свидетельство широты возможностей jQuery и простоты их использования, а также дальновидности разработчиков, предоставивших точки подключения расширений к библиотеке. Я не смогу охватить все расширения в этой книге, но в следующих разделах покажу уровень имеющихся возможностей.

1.3.1. jQuery UI

Проект jQuery UI (http://jqueryui.com/) основан на ядре библиотеки jQuery и представляет собой коллекцию расширений. В их число входит несколько виджетов, включая Tabs, Datepicker и Dialog (см. рис. 1.1), а также реализация разнообразных динамических особенностей пользовательского интерфейса, таких как Draggable и Droppable. Кроме того, в рамках проекта реализовано несколько анимационных эффектов, используемых для отображения/сокрытия элементов и привлечения внимания к ним.

Библиотека jQuery UI реализует собственную инфраструктуру виджетов, образующую единый фундамент для ее компонентов поль-

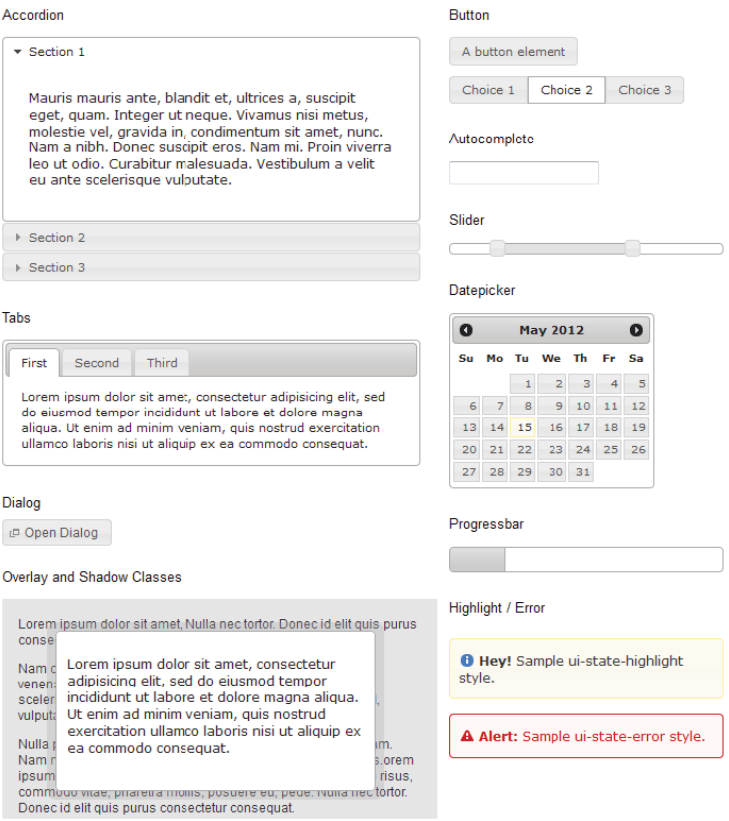


Рис. 1.1 ❖ Примеры виджетов и стилей jQuery UI

зовательского интерфейса. Инфраструктура управляет созданием и уничтожением виджетов, обеспечивает хранение информации об их состоянии и поддерживает взаимодействия с мышью. В главах 8 и 9 мы займемся исследованием этой инфраструктуры и узнаем, как создавать собственные виджеты на ее основе.

Проект объединяет компоненты пользовательского интерфейса и реализацию их динамических особенностей с помощью инструмента ThemeRoller (http://jqueryui.com/themeroller/), упрощающего создание тем, определяющих внешний вид всех виджетов.

Проект jQuery UI сопровождают многочисленные демонстрационные примеры и исчерпывающая документация, позволяющие оценить и использовать все возможности библиотеки. Благодаря модульной архитектуре можно создавать сокращенные версии библиотеки, включающие только те части, которые действительно необходимы. Полный пакет библиотеки вместе с комплектом стандартных тем можно скачасть с любого из сайтов в сети доставки содержимого.

1.3.2. Проверка данных

Как упоминалось выше, для проверки данных на стороне клиента используется расширение Validation[1], написанное Йерном Зафферером (Jörn Zaefferer), как показано на рис. 1.2. Оно упрощает привязку правил проверки к элементам, управление их состоянием и соответствующими им сообщениями об ошибках. Цель расширения состоит в том, чтобы обеспечить ненавязчивую проверку данных, – сообщения об ошибках выводятся только при попытке отправки формы или изменения поля.

Правила могут встраиваться в элементы ввода, в виде значений атрибутов, в коде – для именованных элементов или посредством функций, включаемых в цепочку выбора. Имеется множество предопределенных правил, таких как `required`, `digits`, `date`, `email` и `url`. Некоторые правила могут принимать дополнительные параметры, определяющие их поведение, такие как `minlength` и `maxlength`. Правила могут быть зависимыми от состояния других элементов на странице.

Этот плагин имеет свою точку расширения, дающую возможность определять собственные правила проверки, которые затем будут применяться к указанным элементам наравне с предопределенными. Порядок определения таких правил описывается в главе 14.

---

[1] Домашняя страница расширения Validation: http://jqueryvalidation.org/.

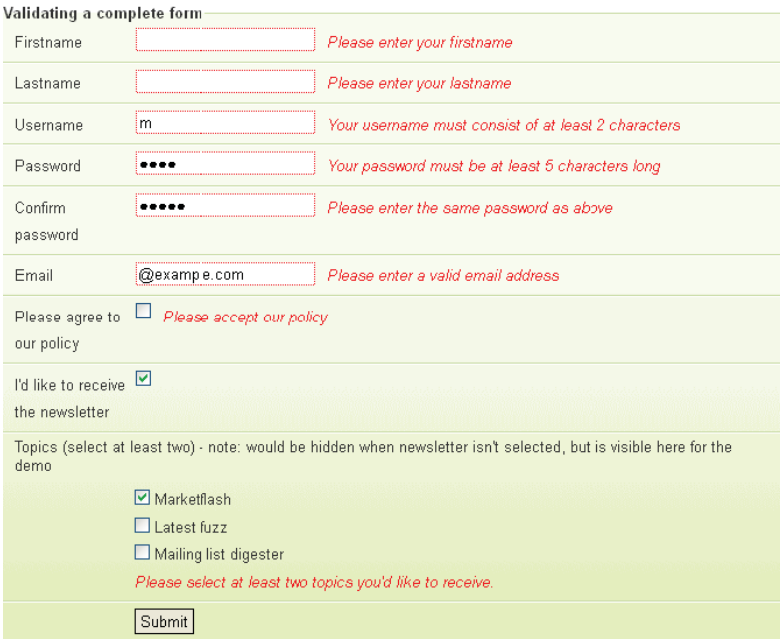


Рис. 1.2 ❖ Расширение Validation в действии. Расширение отображает различные сообщения (наклонным шрифтом) рядом с соответствующими полями, описывающие проблемы, обнаруженные во время проверки

Каждому правилу ставится в соответствие сообщение об ошибке. Сообщения могут переопределяться или переводиться на любой из более чем 30 языков, включенных в пакет. Позицией вывода и группировкой сообщений об ошибках можно управлять посредством параметров функции инициализации.

В комплекте с расширением поставляются исчерпывающая документация и множество примеров. По многочисленным отзывам, Validation – это не только продуманное и хорошо документированное, но и весьма полезное расширение.

1.3.3. Графический ползунок

Расширения могут улучшать представление веб-страницы, отображая содержимое разными, более привлекательными способами. Например, расширение Nivo Slider (http://nivo.dev7studios.com/) пре-

образует простой список изображений в слайд-шоу с различными эффектами, сопутствующими смене изображений.

Привлекательное оформление, изображенное на рис. 1.3, является результатом применения расширения Nivo Slider к разметке HTML, представленной в листинге 1.3. Несмотря на использованные здесь настройки по умолчанию, результат выглядит очень даже неплохо. Как вы уже, наверное, догадались, расширение имеет множество параметров настройки, позволяющих изменять внешний вид и поведение расширения.

Листинг 1.3. Разметка HTML для графического ползунка

```
<div class="slider-wrapper">
    <div id="slider" class="nivoSlider">
        <img src="images/slide1.jpg" alt="" />
        <img src="images/slide2.jpg" alt=""
             title="You can add captions too..." />
        <img src="images/slide3.jpg" alt="" />
    </div>
</div>
```

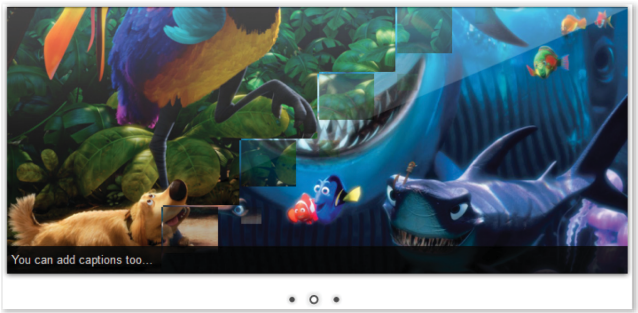


Рис. 1.3 ❖ Расширение Nivo Slider в действии

1.3.4. Интеграция со службой Google Map

Некоторые расширения являются обертками существующих веб-служб, упрощая доступ к ним или скрывая несовместимость разных типов браузеров. Одним из примеров таких расширений может служить плагин gMap (http://gmap.nurtext.de/), обеспечивающий интеграцию с веб-службой Google Map. Несмотря на возможность использования Google Map посредством собственного JavaScript API, расширения, подобные этому, предоставляют более простой интерфейс, инкапсулируя всю необходимую функциональность.

Карта, изображенная на рис. 1.4, была получена с помощью программного кода в листинге 1.4, демонстрирующего простоту использования расширения.

Листинг 1.4. Добавление интеграции со службой Google Map

```
$('#map').gMap({zoom: 4,
    markers: [{address: 'Brisbane, Australia',
               html: 'Brisbane, Australia', popup: true}]
});
```

Рис. 1.4 ❖ Интеграция службы Google Map с помощью расширения gMap

1.3.5. Cookies

Расширение Cookie (https://github.com/carhartl/jquery-cookie) упрощает работу с cookies. Это расширение отличается от представленных выше тем, что его функциональность не применяется к каким-то определенным элементам веб-страницы, а реализована в виде вспомогательной функции, обеспечивающей операции с cookies для всей страницы.

Для создания cookie достаточно просто указать имя и значение:

```
$.cookie('introShown', true);
```

Для настройки cookie функции можно передавать дополнительные параметры – срок действия (по умолчанию срок действия cookie

ограничивается продолжительностью текущего сеанса), имя домена и путь, к которому он применяется, требуется ли для передачи cookie устанавливать защищенное соединение и закодировано ли содержимое cookie.

```
$.cookie('introShown', true, {expires: 30, path: '/'});
```

Чтобы получить значение cookie, достаточно указать только имя. Если в момент вызова cookie с указанным именем не существует, возвращается значение `null`.

```
var introShown = $.cookie('introShown');
```

Чтобы удалить cookie, нужно установить его значение равным `null`.

```
$.cookie('introShown', null);
```

Расширение Cookie подробно рассматривается в главе 6.

1.3.6. Анимация, основанная на изменении цвета

Ядро библиотеки jQuery включает возможность воспроизведения анимационных эффектов, основанных на изменении атрибутов элементов, хранящих простые числовые значения. Для обработки значений атрибутов в любых других форматах требуется реализовать специальный обработчик. С помощью функций из модуля Effects в проекте jQuery UI (http://jqueryui.com) можно реализовать воспроизведение анимационных эффектов, основанных на изменении цвета (http://jqueryui.com/animate/), который может определяться шестнадцатеричным значением (`#DDFFE8` или `#DFE`), значением в формате RGB (`rgb(221, 255, 232)` или `rgb(86%, 100%, 91%)`) или названием (`lime`).

После преобразования значения цвета из того или иного формата в обобщенный формат каждая составляющая цвета (красная/зеленая/синяя) изменяется отдельно, от начального до конечного значения. Реализовав такую возможность в виде расширения, вы сможете использовать стандартную функциональность библиотеки jQuery:

```
$('#myDiv').animate({backgroundColor: '#DDFFE8'});
$('#myDiv').animate({width: 200, backgroundColor: '#DFE'});
```

Подробнее об анимационных эффектах рассказывается в главе 11.

Это нужно знать

Библиотека jQuery является самой используемой библиотекой JavaScript в Сети.

Библиотека jQuery реализует лишь наиболее типичные и часто используемые функциональные возможности, но она поддерживает возможность расширения разными способами.
На основе jQuery сторонними разработчиками создано множество расширений.
Возможности расширения ограничиваются только вашей фантазией.

1.4. В заключение

jQuery выросла до самой используемой библиотеки JavaScript в Сети. Она обладает множеством встроенных возможностей, но главная ее цель – обеспечить базовую инфраструктуру для создания веб-сайтов. Понимая, что библиотека не в состоянии покрыть все возможные потребности, разработчики предусмотрели в ней множество точек расширения, посредством которых другие могут добавлять необходимые им функции.

Вы можете добавлять свои расширения практически в любую часть библиотеки jQuery, от определения собственных селекторов до анимации нечисловых значений атрибутов, создания новых событий и реализации полноценных визуальных компонентов пользовательского интерфейса. Единственное ограничение – ваша фантазия.

Оформление своего кода в виде плагина позволит повторно использовать его в любых других ваших проектах и снизить затраты на тестирование и сопровождение, так как у вас будет лишь одна, общая копия сценария.

В следующей главе вы узнаете, насколько легко расширяется библиотека jQuery, создав простое расширение перед погружением в архитектуру более сложных расширений.

Глава 2

Первое расширение

Эта глава охватывает следующие темы:

- архитектура jQuery;
- создание простого расширения коллекций.

jQuery – это библиотека JavaScript, упрощающая взаимодействие с элементами веб-страниц. Обычно с ее помощью отыскиваются элементы либо прямым выбором, либо путем обхода дерева DOM, и затем к ним применяются некоторые операции. С помощью jQuery можно манипулировать элементами – создавать или удалять их, изменять значения атрибутов и свойств – и добавлять к ним обработчики событий, откликающиеся на действия пользователя. К элементам можно применять анимационные эффекты, изменяющие значения их свойств в течение некоторого интервала времени. Кроме того, jQuery поддерживает технологию Ajax, позволяя выполнять запросы к серверу и получать дополнительную информацию, не разрушая текущего содержимого страницы.

В предыдущей главе я упоминал, что jQuery может далеко не все, поэтому она предоставляет возможность расширения через множество точек интеграции, что привело к бурному росту количества расширений, создаваемых сообществом.

В этой главе сначала рассматривается архитектура jQuery, позволяющая расширениям действовать наравне со встроенным кодом, а затем будет представлено простое расширение коллекций (одно из тех, что воздействуют на множество выбранных элементов), чтобы продемонстрировать доступные возможности. В остальных главах подробно исследуются все точки интеграции, описывается порядок их использования для увеличения возможностей jQuery и формулируется свод правил разработки расширений.

2.1. Архитектура jQuery

Для удобства разработки исходный код jQuery разбит на несколько файлов, однако в процессе сборки они объединяются в единственный

файл, который затем минифицируется для последующей передачи в эксплуатацию либо оставляется в исходном виде для тестирования и отладки. В каждом файле сосредоточен код, реализующий определенный аспект функциональности jQuery, и некоторые из них имеют точки интеграции, дающие другим разработчикам возможность встраивать свои расширения.

Точка интеграции – это атрибут или функция внутри jQuery, где можно зарегистрировать новую функциональность или определенный тип (такой как коллекция функций или расширение Ajax), которые можно интерпретировать подобно стандартным особенностям. Когда в библиотеке возникает необходимость обратиться к расширению, она выполняет обратные вызовы в его код.

На рис. 2.1 изображена схема взаимосвязей между файлами, или модулями, входящими в состав jQuery.

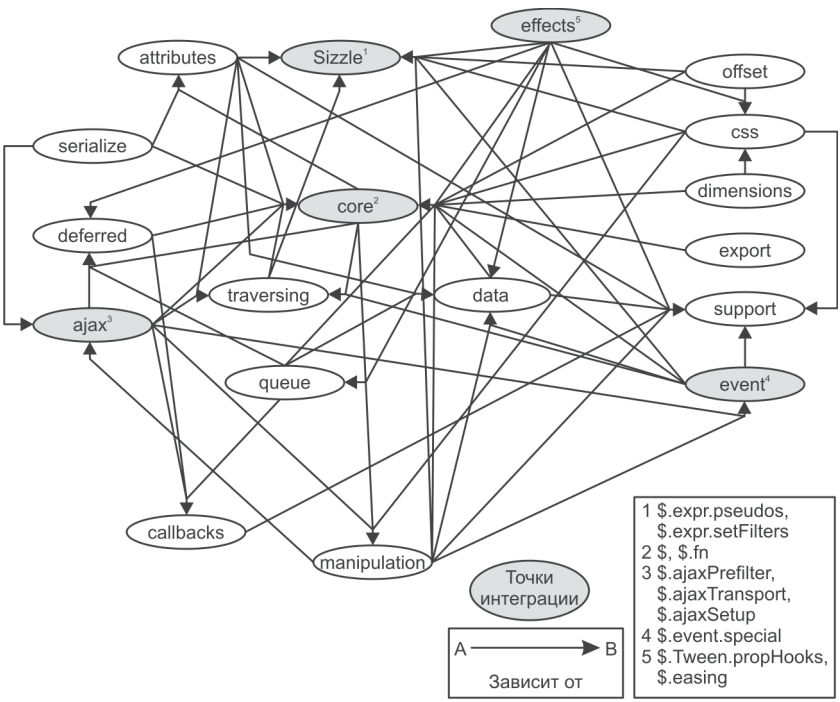


Рис. 2.1 ❖ Модули jQuery, взаимосвязи между ними и точки интеграции расширений

К числу основных модулей, поддерживающих возможность расширения jQuery (закрашены на рис. 2.1), относятся: библиотека *Sizzle*, реализующая функции выбора элементов из дерева DOM; модуль *core*, включающий саму функцию `jQuery`; модуль *ajax*, реализующий поддержку технологии Ajax; модуль *event*, реализующий механизм событий; и модуль *effects*, реализующий механизм анимации.

2.1.1. Точки интеграции с библиотекой jQuery

Все имеющиеся точки интеграции с библиотеками jQuery и jQuery UI перечислены в табл. 2.1 и описываются в следующих разделах. Напомню, что идентификатор `$` – это псевдоним функции `jQuery` (если он не был освобожден вызовом функции `noConflict`).

Таблица 2.1. Точки интеграции с библиотекой jQuery

Точка интеграции	Назначение	Примеры	Где обсуждается
`$`	Вспомогательные функции	`$.trim` `$.parseXML`	Глава 6
`$.ajaxPrefilter`	Предварительные фильтры Ajax	`$.ajaxPrefilter('script', ...)`	Глава 12
`$.ajaxSetup`	Преобразователи типов данных Ajax	`$.ajaxSetup({converters: {'text xml', $.parseXML}})`	Глава 12
`$.ajaxTransport`	Транспортные механизмы Ajax	`$.ajaxTransport('script', ...)`	Глава 12
`$.easing`	Функции управления переходами при воспроизведении анимации	`$.easing.swing` `$.easing.easeOutBounce`	Глава 10
`$.effects`	Визуальные эффекты jQuery UI (jQuery UI 1.8-)	`$.effects.clip` `$.effects.highlight`	Глава 10
`$.effects.effect`	Визуальные эффекты jQuery UI (jQuery UI 1.9+)	`$.effects.effect.clip` `$.effects.effect.highlight`	Глава 10
`$.event.special`	Нестандартные события	`$.event.special.mouseenter` `$.event.special.submit`	Глава 13

`$.expr.filters`
`$.expr[':']`
`$.expr.setFilters`
	Селекторы (jQuery 1.7-)	`$.expr.filters.hidden`
		`$.expr.setFilters.odd`
			Глава 3

`$.expr.pseudos`
`$.expr[':']`
`$.expr.setFilters`
	Селекторы (jQuery 1.8+)	`$.expr.pseudos.enabled`
		`$.expr.setFilters.first`
			Глава 3

`$.fn`	Расширения коллекций	`$.fn.show` `$.fn.append`	Глава 5
`$.fx.step`	Анимация атрибутов (jQuery 1.7-)	`$.fx.step.opacity`	Глава 11

Таблица 2.1 (окончание)

Точка интеграции	Назначение	Примеры	Где обсуждается
`$.Tween.propHooks`	Анимация атрибутов (jQuery 1.8+)	`$.Tween.propHooks.scrollTop`	Глава 11
`$.validator.addMethod`	Правила для расширения Validation	`$.validator.addMethod( 'USPhone', ..., ...)`	Глава 14
`$.widget`	Виджеты jQuery UI	`$.widget('ui.tabs', ...)`	Главы 8, 9

2.1.2. Селекторы

В состав библиотеки jQuery входит механизм Sizzle выбора элементов. Это самостоятельная библиотека, реализующая поддержку выбора и позволяющая находить требуемые элементы внутри веб-страницы. Всегда, когда это возможно, она делегирует выполнение операций функциям браузера, чтобы добиться максимальной скорости работы. Операции, которые не могут быть выполнены браузером, выполняются программным кодом на JavaScript. Например, чтобы найти все элементы `label`, следующие сразу за элементами `input` (например, метки флажков), в пределах элемента с атрибутом `id="preferences"`, можно использовать вызов:

```
$('#preferences input + label')...
```

Библиотека Sizzle дает возможность выбирать элементы по именам узлов, по значениям атрибутов `id` и `class` или по признаку вложенности. Можно также использовать селекторы псевдоклассов, включая определяемые спецификацией каскадных таблиц стилей (Cascading Style Sheets, CSS), и другие, добавляемые самой библиотекой Sizzle, такие как `:checked`, `:even` и `:not`. Комбинируя несколько селекторов в одной строке с условием выбора, можно отбирать только те элементы, которые действительно необходимы.

Селекторы псевдоклассов

Согласно спецификации CSS: «Псевдоклассы классифицируют элементы не по их именам, атрибутам или содержимому, а по другим характеристикам; по характеристикам, которые в принципе не могут быть получены из дерева документа»[1]. Эти селекторы начинаются с двоеточия (`:`) и включают определение условия выбора: по позиции (`:nth-child(n)`), по содержимому (`:empty`) и по отрицанию (`:not(selector)`).

---

[1] W3C, Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, «5.10 Pseudo-elements and pseudo-classes», http://www.w3.org/TR/2011/REC-CSS2-20110607/selector.html#pseudo-elements.

Вы можете добавлять собственные селекторы псевдоклассов и интегрировать их в процесс выбора с помощью точки интеграции `$.expr.pseudos` (или `$.expr.filters` в версиях jQuery ниже 1.8). В конечном счете селектор – это обычная функция, которая получает элемент в аргументе и возвращает `true`, если элемент соответствует селектору, и `false` в противном случае.

Расширения, добавленные через точку интеграции `$.expr.setFilters`, могут фильтровать элементы, опираясь на их позиции в текущем множестве совпавших элементов. Расширение должно быть реализовано в виде функции, возвращающей отфильтрованное множество элементов (в версии jQuery 1.8 и выше) или логический флаг, определяющий необходимость включения (в версии jQuery 1.7 и ниже).

Подробнее о добавлении собственных селекторов и фильтров в jQuery/Sizzle рассказывается в главе 3.

2.1.3. Расширения коллекций

Расширения коллекций выполняют операции с множествами элементов, полученными в результате процедуры выбора или последовательного обхода дерева DOM. К тому же они являются одними из распространенных расширений jQuery.

Для внедрения в библиотеку jQuery эти расширения должны подключаться к точке интеграции `$.fn` в виде функций, реализующих их возможности. Если заглянуть в реализацию библиотеки, можно заметить, что `$.fn` является простым псевдонимом для `$.prototype`. То есть любые функции, подключаемые к данной точке интеграции, будут доступны в любых объектах-коллекциях jQuery, полученных в результате вызова функции `jQuery` с селектором или элементами DOM. Они могут вызываться относительно этих коллекций в соответствующем контексте.

Все расширения коллекций должны возвращать текущий набор элементов или новый, если они реализуют какие-либо преобразования, чтобы их можно было включать в цепочки вызовов вместе с другими методами jQuery, – эта особенность является ключевой парадигмой выполнения операций в jQuery.

Основные принципы разработки расширений будут представлены в главе 4, а в главе 5 описывается реализующая эти принципы инфраструктура поддержки расширений, которую можно использовать для создания новых расширений коллекций.

2.1.4. Вспомогательные функции

Функции, не предназначенные для работы с коллекциями выбранных элементов (такие как встроенные функции `trim` и `parseXML`), могут внедряться в мир jQuery непосредственно через точку интеграции `$`. В действительности внедрять вспомогательные функции необязательно (их можно определить как обычные автономные функции), но они часто используют другие возможности jQuery, и подобным внедрением достигается единообразие использования библиотеки jQuery. Включение их в библиотеку также помогает уменьшить захламление глобального пространства имен, снизить вероятность конфликтов и позволяет хранить родственные функции в одном месте.

Вспомогательные функции не ограничиваются фиксированным списком параметров и могут принимать все, что потребуется для выполнения реализуемых ими операций.

Приемы добавления новых вспомогательных функций рассматриваются в главе 6.

2.1.5. Виджеты jQuery UI

Библиотека jQuery UI является официальной коллекцией компонентов пользовательского интерфейса, поведенческих особенностей и эффектов, реализованных на основе базовой библиотеки jQuery. Она предоставляет инфраструктуру поддержки виджетов, реализующую основные принципы создания новых компонентов или эффектов.

Виджеты создаются вызовом функции `$.widget` из библиотеки jQuery UI. Эта функция принимает имя нового виджета (включающее пространство имен, чтобы избежать конфликтов), необязательную ссылку на базовый «класс», наследуемый виджетом, и коллекцию функций, расширяющих и переопределяющих базовые возможности. Инфраструктура виджетов управляет их применением к выбранным элементам; устанавливает, извлекает и сохраняет параметры, определяющие внешний вид и поведение виджетов; а также удаляет виджеты, когда они становятся ненужными. Отображение функций, вызываемых пользователем, в методы, определяемые виджетом, выполняется внутренней функцией `$.widget.bridge`.

Подробнее об инфраструктуре поддержки и о виджетах jQuery UI рассказывается в главе 8. В главе 9 исследуются приемы использования модуля Mouse из библиотеки jQuery UI для реализации новых компонентов, использующих возможности взаимодействия с указателем мыши.

2.1.6. Эффекты jQuery UI

Еще одной важной составляющей библиотеки jQuery UI являются эффекты для анимации элементов веб-страниц. Большинство этих эффектов применяется для реализации постепенного сокрытия или появления элементов, как, например, `blind` и `drop`, но некоторые, такие как `highlight` и `shake`, служат для привлечения внимания пользователя к определенным элементам.

Внедрять новые эффекты в библиотеку jQuery UI можно с помощью точки интеграции `$.effects.effect` (или `$.effects`, в версиях jQuery UI 1.8 и ниже). После внедрения эти эффекты будут доступны (по именам) для использования с функциями `effect`, `show`, `hide` или `toggle` из библиотеки jQuery UI. Каждый эффект является функцией, добавляющей функцию обратного вызова в очередь `fx` текущего элемента и реализующей анимационный эффект, который будет воспроизведен после обработки предыдущего эффекта, включенного в очередь ранее.

Функции управления переходами (easings) определяют, как будет изменяться значение атрибута с течением времени, и могут применяться к анимационным эффектам для управления ускорением и замедлением изменений. Несмотря на то что функции переходов являются частью jQuery, библиотекой предлагаются только две такие функции – `linear` и `swing`. Библиотека jQuery UI добавляет еще 30 функций переходов. Добавление собственных функций переходов осуществляется с помощью точки интеграции `$.easing`. Эти функции должны возвращать величину изменения значения атрибута (в нормализованном виде – от 0 до 1) с учетом прошедшего времени с начала воспроизведения анимационного эффекта (также нормализованного на интервале от 0 до 1).

Подробнее о функциях переходов и эффектах jQuery UI рассказывается в главе 10.

2.1.7. Анимация свойств

Поддержка анимации в jQuery дает возможность изменять значения различных атрибутов выбранных элементов. Обычно эти атрибуты определяют внешний вид, в результате их изменения возникают эффекты перемещения элементов, изменение их размеров или рамок, а также изменение размеров шрифтов, которыми отображается содержимое элементов. Но библиотека jQuery поддерживает возможность изменения только простых числовых значений атрибутов (включая определение единиц измерения). Для анимации атрибутов

со сложными значениями необходимо добавлять собственные функции анимации.

Подключение новых механизмов анимации атрибутов со сложными значениями осуществляется через точку интеграции `$.Tween.propHooks` путем передачи двух функций – функции чтения и функции записи значения атрибута. В версиях jQuery 1.7 и ниже используется точка интеграции `$.fx.step`, которой должна передаваться одна функция, реализующая один шаг анимации.

Подробнее о добавлении новых анимационных эффектов рассказывается в главе 11.

2.1.8. Поддержка Ajax

Поддержка технологии Ajax является одной из важнейших особенностей библиотеки jQuery. Она упрощает получение информации со стороны сервера и ее встраивание в текущую страницу без необходимости полностью обновлять ее. Поскольку серверы могут поставлять данные в разных форматах, jQuery определяет множество *предварительных фильтров*, управляющих процессом извлечения данных; *транспортов*, обеспечивающих собственно извлечение данных; и *преобразователей*, выполняющих преобразование полученных данных в требуемый формат. Каждый из этих механизмов может расширяться под определенные требования. На рис. 2.2 показано, как выглядит стандартная процедура выполнения запроса Ajax.

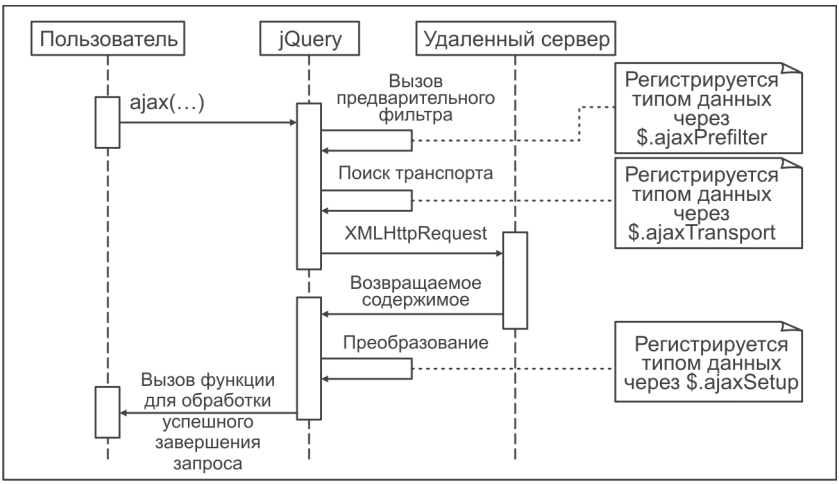


Рис. 2.2 ❖ Диаграмма стандартной процедуры выполнения запроса Ajax с точками интеграции

Вызовом функции `$.ajaxPrefilter` можно зарегистрировать собственную функцию, которая будет вызываться при попытке выполнить запрос данных в указанном формате, таком как `html`, `xml` или `script`. Этой функции передается ссылка на объект `XMLHttpRequest`, который будет использоваться для выполнения запроса, благодаря чему вы сможете настроить параметры запроса и даже отменить его.

Вызовом функции `$.ajaxTransport` можно зарегистрировать функцию, реализующую фактическое извлечение данных указанного формата. Эта возможность позволяет настроить доступ к данным (например, загрузить изображение непосредственно в элемент `Image`). По умолчанию используется объект `XMLHttpRequest`.

Обычно данные возвращаются сервером в текстовом формате, но иногда бывает желательно преобразовать их в некоторый другой формат, например превратить разметку XML в дерево DOM. Вызовом функции `$.ajaxSetup` можно зарегистрировать функцию, преобразующую исходные данные в другой формат, которые затем будут возвращены как результат выполнения запроса Ajax.

Подробнее об определении новых предварительных фильтров, транспортов, протоколов и преобразователей рассказывается в главе 12.

2.1.9. Обработка событий

Библиотека jQuery позволяет подключать к выбранным элементам обработчики событий, которые будут вызываться в ответ на действия пользователя. Эта возможность часто используется для обработки событий от мыши или клавиатуры, а также событий изменения состояния. При необходимости можно определить собственные события для обработки нестандартных ситуаций.

Регистрация новых событий выполняется вызовом функции `$.event.special`. При этом в вызов функции передается тип события, а также функции настройки событий, удаления поддержки событий, когда она станет не нужна, и возбуждения событий при наступлении соответствующих условий.

Подробнее о создании и обработке собственных событий рассказывается в главе 13.

2.1.10. Правила проверки данных

Несмотря на то что расширение Validation не является частью ядра библиотеки jQuery, оно используется достаточно часто и предоставляет свою точку интеграции, посредством которой можно регистри-

ровать собственные правила проверки данных. Эти правила можно использовать наряду со встроенными, такими как `required` и `number`, и проверять с их помощью наличие и правильность данных в полях форм перед отправкой на сервер.

Регистрация собственных правил осуществляется вызовом функции `$.validator.addMethod`, которой передается имя правила; функция, возвращающая `true`, если элемент содержит допустимое значение, и `false` – в противном случае; и текст сообщения об ошибке. Для автоматического включения нового правила через атрибут `class` отдельного элемента можно воспользоваться функцией `$.validator.addClassRules`.

Подробнее об определении собственных правил проверки рассказывается в главе 14.

2.2. Простое расширение

Расширения для библиотеки jQuery могут делать практически все, что угодно, о чем свидетельствует широкий выбор имеющихся сторонних расширений. Это могут быть простые расширения, воздействующие на отдельные элементы или сложные, изменяющие внешний вид и поведение множества элементов, как, например, расширение Validation.

Чаще других создаются расширения коллекций, добавляющие новые функциональные возможности к коллекциям элементов, полученным в результате выполнения процедуры выбора или последовательного обхода дерева DOM. Простым примером расширений этого типа может служить расширение Watermark, реализующее отображение подсказки в поле ввода. Исследование процедуры создания такого расширения поможет вам получить более полное представление о том, как создаются расширения.

2.2.1. Текст подсказки

Чтобы сэкономить место внутри формы, метки полей ввода иногда опускаются и замещаются текстом подсказки (внутри самого поля ввода), который исчезает, когда поле получает фокус ввода. Если поле останется незаполненным, в нем вновь выводится текст подсказки. Чтобы не вводить пользователя в заблуждение и показать, что этот текст не является фактическим значением поля ввода, он обычно выводится серым цветом. Такие подсказки часто называют *водяными знаками* (watermark).

Текст подсказки можно было бы определять на этапе инициализации расширения, но лучше определять его отдельно для каждого поля и тем самым обеспечить отдельные настройки для каждого поля в отдельности. Для хранения текста подсказки идеально подходит атрибут `title`. Этот атрибут специально предназначен для хранения короткого описания поля, и его содержимое может отображаться браузером в виде всплывающей подсказки при наведении указателя мыши на поле ввода. Для слабовидящих это расширение может служить дополнительной подсказкой о том, в каком поле находится фокус ввода.

Когда поле ввода получает фокус ввода, необходимо удалить текст подсказки, если он присутствует, и изменить стиль отображения поля, чтобы оно выглядело как активное. Аналогично, когда поле теряет фокус ввода, необходимо восстановить в нем текст подсказки и установить неактивный стиль отображения, если поле осталось пустым. На рис. 2.3 изображено действие расширения Watermark в разные моменты ввода данных.

Рис. 2.3 ❖ Действие расширения Watermark: перед взаимодействием, после ввода имени и в готовности к отправке

Для большей гибкости визуальное оформление полей при отображении текста подсказки должно контролироваться с помощью CSS. При отображении подсказки можно присваивать полю некоторый класс CSS и удалять его при получении полем фокуса ввода или когда в нем присутствует текст, введенный пользователем. В этом случае фактическое визуальное оформление перекладывается на стили CSS и легко может быть переопределено пользователем.

2.2.2. Реализация расширения Watermark

Так как расширение применяется к элементам страницы, оно является расширением коллекций, то есть воздействует на коллекцию элементов, найденных в результате выбора по селектору или последо-

вательным обходом дерева DOM. Отсюда следует, что подключаться это расширение должно к точке интеграции `$.fn`.

В листинге 2.1 приводится полная реализация расширения Watermark.

Листинг 2.1. Расширение Watermark

```
// ❶ Объявление функции расширения
$.fn.watermark = function(options) {
    // ❷ Параметры настройки
    options = $.extend({watermarkClass: 'watermark'},
                       options || {});
    // ❸ При получении фокуса...
    return this.focus(function() {
        var field = $(this);
        // ❹ ...если содержит подсказку...
        if (field.val() == field.attr('title')) {
            // ❺ ...удалить подсказку
            field.val('').
                removeClass(options.watermarkClass);
        }
    // ❻ При потере фокуса ввода
    }).blur(function() {
        var field = $(this);
        // ❼ ...если пустое...
        if (field.val() == '') {
            // ❽ ...восстановить подсказку
            field.val(field.attr('title')).
                addClass(options.watermarkClass);
        }
    }).blur(); // ❾ Инициализировать поле
};
```

Расширение подключается к точке интеграции `$.fn` объявлением функции `$.fn.watermark` ❶, что обеспечивает возможность его внедрения в обработку выбираемых множеств элементов. Новый атрибут получает имя особенности, которую он реализует, и становится доступен по этому имени. Значением атрибута является функция, принимающая единственный аргумент (`options`) и добавляющая новые особенности к целевым полям. В аргументе функция ожидает получить единственный параметр настройки: имя класса CSS, используемого для оформления внешнего вида поля, чтобы показать, что оно содержит текст подсказки. Для этого параметра предусмотрено значение по умолчанию, которое может быть переопределено пользователем ❷. Значение по умолчанию определяется как объект с атрибутом `watermarkClass` и значением `'watermark'`, который расширяется другими параметрами настройки, возможно, переопределяющими значе-

ния по умолчанию. Конструкция || {} гарантирует замену аргумента options пустым объектом, если он не был передан пользователем.

Чтобы обеспечить возможность включения этого расширения в цепочку вызовов других расширений, что является одним из основных элементов философии jQuery, функция должна возвращать множество элементов, над которыми выполняется операция ❸. Так как большинство встроенных функций jQuery также возвращают тот же самый набор, можно просто вернуть результат вызова стандартной функции jQuery.

К каждому полю в выбранном множестве добавляется обработчик события focus ❸. Внутри этого обработчика сохраняется ссылка на текущее поле, представленное объектом jQuery, потому что она еще будет использоваться неоднократно. Затем текущее содержимое поля сравнивается со значением его атрибута title ❹. Если они равны, текст подсказки замещается пустой строкой ❺, и из атрибута class удаляется класс, указанный в параметрах настройки.

Так как работа продолжается с тем же набором полей и со всеми ними выполняются одни и те же операции, мы можем добавить в цепочку дополнительный обработчик blur ❻. И снова обработчик сохраняет ссылку на текущее поле для последующего использования. На этот раз он сравнивает текущее значение поля с пустой строкой ❼ и в случае выполнения условия восстанавливает текст подсказки (из атрибута title) и добавляет класс CSS ❽.

В заключение необходимо вызвать только что определенный обработчик blur ❾, чтобы инициализировать поле в зависимости от его текущего значения.

2.2.3. Удаление текста подсказок

Так как строки подсказок устанавливаются в качестве значений всех полей, они могут быть отправлены на сервер и обработаны им, если не предусмотреть некоторые дополнительные операции. Подсказки – это артефакты пользовательского интерфейса, поэтому перед отправкой формы их следует удалить.

Самый простой способ реализовать удаление подсказок – определить еще одно расширение коллекций, выполняющее эту задачу. В листинге 2.2 представлена эта дополнительная функция.

Листинг 2.2. Удаление подсказок из значений полей

```
// ❶ Объявление функции расширения
$.fn.clearWatermark = function() {
```

```
    // ❷ Для каждого поля...
    return this.each(function() {
        // ❸ ...если содержит подсказку...
        var field = $(this);
        if (field.val() == field.attr('title')) {
            // ❹ ...удалить ее
            field.val('');
        }
    });
};
```

И снова расширение подключается к точке интеграции `$.fn` в виде функции `$.fn.clearWatermark`, потому что данное расширение так же выполняет операции с множеством элементов страницы ❶. Оно так же обеспечивает возможность включения в цепочки вызовов, возвращая ссылку на текущее множество выбранных элементов обращением к функции `each` ❷. Для каждого выбранного элемента сравнивается его текущее значение с атрибутом `title` ❸, и если они равны, в значение поля записывается пустая строка ❹.

Эта функция должна вызываться непосредственно перед отправкой формы на сервер, в противном случае текст подсказок будет участвовать в дальнейшей обработке. При необходимости вы легко сможете восстановить подсказки во всех полях, вызвав обработчик `blur`.

2.2.4. Применение расширения Watermark

Чтобы задействовать расширение Watermark, поместите предыдущий код непосредственно в веб-страницу или сохраните его в отдельном файле JavaScript (jquery.watermark.js) и подключите этот файл к странице. Аналогично для оформления внешнего вида расширения можно добавить соответствующие стили непосредственно в страницу или подключить их, если они хранятся в отдельном файле CSS (jquery.watermark.css). Если сохранить расширение и стили в отдельных файлах, это упростит повторное их использование в других страницах.

На рис. 2.4 показано, как выглядит страница, демонстрирующая применение расширения Watermark, а в листинге 2.4 приводится ее код разметки.

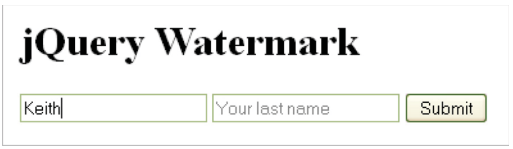


Рис. 2.4 ❖ Страница, демонстрирующая применение расширения Watermark

Листинг 2.3. Применение расширения Watermark

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

<!-- ❶ Загрузить стили для расширения Watermark -->
<link type="text/css" href="jquery.watermark.css" rel="stylesheet">

<!-- ❷ Загрузить библиотеку jQuery -->
<script type="text/javascript"
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.
js">
</script>

<!-- ❸ Загрузить расширение Watermark -->
<script type="text/javascript" src="jquery.watermark.js"></script>
<script> type="text/javascript"
// ❹ Когда дерево DOM будет загружено...
$(function() { // Сокращение для $(document).ready(function() {
    // ❺ ...применить расширение Watermark
    $('input.wmark').watermark();
    // ❻ При отправке на сервер...
    $('#submit').click(function() {
        // ❼ ...удалить подсказки
        $('input.wmark').clearWatermark();
        alert('Welcome ' + $('#first').val() + ' ' + $('#last').val());
        // ❽ ...восстановить подсказки
        $('input.wmark').blur();
    });
});
</script>
</head>
<body>
<h1>jQuery Watermark</h1>
<p>
    <!-- ❾ Поля, к которым применяется расширение -->
    <input type="text" id="first" class="wmark" title="Your first name">
    <input type="text" id="last" class="wmark" title="Your last name">
    <input type="button" id="submit" value="Submit">
</p>
</body>
</html>
```

Сначала страница загружает файл CSS, содержащий стили для расширения Watermark ❶. Применение стилей обеспечивает отображение текста подсказки серым цветом (предполагается, что по умолчанию используется класс CSS):

```
.watermark { color: #888; }
```

Библиотека jQuery должна загружаться первой ❷, чтобы она была доступна для использования расширением, которое загружается следующим ❸. Когда остальная часть дерева DOM загрузится и будет готова к использованию ❹, можно задействовать новое расширение.

Функциональные возможности расширения добавляются к выбранным полям (то есть к элементам `input` с классом `wmark`) вызовом функции расширения ❺. Напомню, что функция возвращает первоначальный набор выбранных элементов, поэтому ее можно включать в цепочки вызовов других функций. Так как поля ввода первоначально пусты, расширение автоматически запишет в них текст подсказки, извлеченный из атрибутов `title`.

Перед отправкой полей ввода на сервер для дальнейшей обработки ❻ необходимо сначала удалить текст подсказки из каждого поля ❼. После отправки можно восстановить текст подсказки в пустых полях, вызвав обработчики `blur` полей ❽.

Фактические поля ввода, к которым применяется расширение Watermark, находятся в теле документа ❾ и помечены классом `wmark`, что упрощает их выбор.

Открыв эту страницу в своем браузере, вы увидите, что первоначально каждое поле содержит текст подсказки с соответствующим оформлением. При передаче фокуса ввода в поле текст подсказки удаляется, и вы можете ввести соответствующее значение. Если покинуть поле без ввода какого-либо значения, текст подсказки восстанавливается, и поле опять оформляется соответствующим стилем. Когда выполняется щелчок на кнопке **Submit (Отправить)**, все подсказки удаляются из полей, и значения полей отображаются в окне диалога, после закрытия которого в пустых полях восстанавливаются подсказки.

Это простое расширение коллекций позволяет получить представление о том, чего можно добиться всего несколькими строками кода. Поместив код в отдельный файл, вы легко сможете повторно использовать его в других страницах, где требуется та же функциональность.

В последующих главах описываются основные принципы разработки расширений, которых следует придерживаться, чтобы сделать свои расширения надежными и практичными. Вы также познакомитесь с двумя инфраструктурами поддержки расширений, реализующими наиболее общие требования, предъявляемые к расширениям, и узнаете, как тестировать, упаковывать и документировать свои расширения, чтобы другие могли использовать их с максимальной пользой для себя.

Это нужно знать

Библиотека jQuery изначально проектировалась с поддержкой расширения.
Библиотека jQuery имеет множество точек интеграции, которые позволяют добавлять в нее функциональные особенности разного типа.
Создать простое расширение совсем несложно.
Добавление расширения коллекций, воздействующего на множество выбранных элементов, производится в точке интеграции `$.fn`.
Сохраняйте код своих расширений в отдельных файлах, чтобы упростить их использование в других страницах.

Попробуйте сами

Измените расширение Watermark так, чтобы оно извлекало текст подсказки из атрибута `data-label`. Это позволит использовать атрибут `title` для организации вывода более длинных пояснений в виде всплывающих подсказок и сохранит возможность восстановления текста подсказки для пустых полей.

2.3. В заключение

Благодаря своей прозорливости разработчики библиотеки jQuery приложили немалые усилия, чтобы реализовать в ней обширную инфраструктуру и разнообразные функциональные возможности, включая точки интеграции, для дальнейшего расширения ее возможностей. Дополнительные расширения внедряются в библиотеку и используются наравне со встроенными возможностями.

В jQuery имеется значительное число точек интеграции, каждая из которых дает возможность расширять определенный аспект библиотеки, от нестандартных селекторов до расширений коллекций, вспомогательных функций, виджетов и эффектов jQuery, инструментов поддержки технологии Ajax и событий. Вы имеете уникальную возможность настроить jQuery под свои нужды. Каждая точка интеграции подробно обсуждается далее в этой книге.

На примере простого расширения коллекций вы увидели, как создать полезное и практичное расширение, написав всего несколько строк кода. Упаковав этот код в отдельный файл, вы сможете повторно использовать его возможности в других веб-страницах.

В следующей главе вы увидите, как добавить в библиотеку собственные селекторы, которые являются одними из самых простых расширений.

Глава 3

Селекторы и фильтры

Эта глава охватывает следующие темы:

- существующие типы селекторов jQuery;
- добавление собственных селекторов псевдоклассов;
- добавление фильтров.

Один из принципов библиотеки jQuery гласит: «выбирай и действуй». Согласно ему вы выбираете один или более элементов и затем выполняете некоторые операции с ними. jQuery включает копию механизма выбора Sizzle (который представляет собой «механизм селекторов CSS, реализованный исключительно на языке JavaScript и предназначенный для включения в основную библиотеку»; http://sizzlejs.com/), выполняющего выбор элементов в соответствии с указанными условиями. Когда это возможно, библиотека Sizzle делегирует операции выбора встроенным механизмам браузера, а при необходимости обрабатывает селекторы самостоятельно.

Несмотря на большое разнообразие встроенных селекторов, иногда бывает желательно создать собственный селектор, который мог бы интегрироваться с другими механизмами jQuery. К счастью, добавление нового селектора – одна из самых простых задач по расширению jQuery.

> Несмотря на то что первоначально механизм выбора Sizzle создавался как составная часть библиотеки jQuery, в настоящее время это отдельный проект, который может использоваться любыми библиотеками JavaScript. Чтобы обеспечить возможность участия в его разработке для всех желающих, код Sizzle был передан под эгиду организации Dojo Foundation (http://dojofoundation.org/; не путайте с Dojo Toolkit). Благодаря этому удалось привлечь заинтересованных разработчиков и из других сообществ в экосистеме JavaScript.

3.1. Что такое селекторы и фильтры?

Селекторы jQuery основаны на селекторах, определяемых спецификациями CSS. Это строковые значения, идентифицирующие элементы дерева HTML DOM. В терминологии jQuery селекторы также называют *фильтрами*. И те, и другие сужают множество элементов, имеющихся в текущем контексте.

3.1.1. Зачем добавлять новые селекторы?

В библиотеке jQuery имеется множество встроенных селекторов, тогда зачем добавлять новые? Если, например, для выбора элементов страницы требуется использовать многоходовую процедуру, и эта процедура используется многократно, либо в одной странице, либо в нескольких, это может служить поводом задуматься о создании собственного селектора, реализующего такой выбор.

Даже если коллекцию элементов можно получить путем комбинирования селекторов и функций jQuery, собственные селекторы могут обеспечивать более краткий и выразительный способ отбора. Выбирая говорящие имена для своих селекторов, можно более ясно обозначить, какой результат получается в ходе выбора, чем это позволяют многоходовые комбинации, способные вносить путаницу. Аналогично вызов собственного кода обычно более краток и оставляет меньше шансов забыть выполнить какую-то промежуточную операцию, потому что процесс выбора имеет только одно определение. Кроме того, код селектора проще тестировать, а любые исправления и дополнения немедленно применяются ко всем страницам, где он используется.

Например, чтобы выбрать элементы списка, кроме первого и последнего, можно было бы использовать следующую комбинацию встроенных селекторов:

```
$('li:not(:first, :last)', list)...
```

Но можно определить собственный селектор и использовать его:

```
$('li:middle', list)...
```

Если требуется отыскать все абзацы с текстом, выделенным некоторым образом, можно было бы использовать такой способ:

```
$('p:has(b | i | em | strong)')...
```

Или, чтобы уберечь себя от ошибок и опечаток, определить собственный селектор:

```
$('p:has(:emphasis)')...
```

В следующих двух подразделах мы коротко рассмотрим селекторы, встроенные в библиотеку jQuery. Затем вы увидите, как определять собственные селекторы.

3.1.2. Простые селекторы

В библиотеке jQuery существует множество простых селекторов, позволяющих выбирать элементы по именам, значениям атрибутов `id` и `class` и/или по имени атрибута и его значению (они перечислены в табл. 3.1). Большинство из них реализовано в виде низкоуровневых функций внутри браузера, благодаря чему они выполняются намного быстрее, чем другие селекторы.

Селекторы могут объединяться для создания более сложных критериев выбора. Например, чтобы найти все элементы `label`, непосредственно следующие за элементами `input` (это могут быть метки для флажков) и находящиеся внутри элемента с атрибутом `id="content"`, можно было бы использовать следующий селектор:

```
$('#content input + label')...
```

Таблица 3.1. Простые селекторы jQuery

Селектор	Шаблон	Назначение
Селектор «все»	`*`	Выбирает все элементы независимо от значения атрибута `name`
Селектор по именам элементов	`element`	Выбирает все элементы с соответствующим именем, например: `input`
Селектор по атрибуту `id`	`#identifier`	Выбирает один элемент с соответствующим значением атрибута `id`, например: `#field1`
Селектор по атрибуту `class`	`.class`	Выбирает все элементы с соответствующим значением атрибута `class`, например: `div.tabs`
Селектор потомков	`parent child`	Выбирает все *дочерние* элементы, содержащиеся на любом уровне вложенности в родительских элементах `parent`, например: `#list li`
Селектор дочерних элементов	`parent > child`	Выбирает все *дочерние* элементы, являющиеся прямыми потомками родительских элементов `parent`, например: `#tabs > div`

Таблица 3.1 (продолжение)

Селектор	Шаблон	Назначение
Селектор смежных элементов, следующих далее	`prev + next`	Выбирает все *последующие* элементы, которым непосредственно предшествует элемент `prev`, например: `input + next`
Селектор братских элементов, следующих далее	`prev ~ siblings`	Выбирает все *братские* элементы, которым непосредственно предшествует элемент `prev` и имеющие общего родителя, например: `h2 ~ p`
Селектор по наличию атрибута	`[name]`	Выбирает все элементы, имеющие указанный атрибут, независимо от его значения, например: `input[readonly]`
Селектор по равенству атрибута некоторому значению	`[name="value"]`	Выбирает все элементы, имеющие указанный атрибут с указанным значением, например: `label[for="field1"]`
Селектор по неравенству атрибута некоторому значению	`[name!="value"]`	Выбирает все элементы, в которых отсутствует указанный атрибут или присутствует, но его значение не равно указанному, например: `a[target!="_blank"]`
Селектор по атрибуту, начинающемуся с некоторого значения	`[name^="value"]`	Выбирает все элементы, имеющие указанный атрибут со значением, начинающимся с заданного, например: `a[href^="http:"]`
Селектор по атрибуту, заканчивающемуся некоторым значением	`[name$="value"]`	Выбирает все элементы, имеющие указанный атрибут со значением, заканчивающимся заданным, например: `a[href$=".pdf"]`
Селектор по атрибуту, содержащему некоторое значение	`[name*="value"]`	Выбирает все элементы, имеющие указанный атрибут со значением, содержащим заданное, например: `a[href*="google"]`
Селектор по атрибуту, содержащему некоторое слово	`[name~="value"]`	Выбирает все элементы, имеющие указанный атрибут со значением, содержащим заданное слово, ограниченное пробелами, например: `a[title~="Google"]`
Селектор по атрибуту, содержащему некоторый префикс	`[name|="value"]`	Выбирает все элементы, имеющие указанный атрибут со значением, равным заданному или начинающимся с заданного, за которым дополнительно следует дефис, например: `a[class|="ui-state"]`

Таблица 3.1 (окончание)

Селектор	Шаблон	Назначение
Селектор по множеству атрибутов	`[name1="value1"][name2="value2"]`	Выбирает все элементы, имеющие все указанные атрибуты с указанными значениями, например: `input[type="checkbox"][disabled]`
Селектор по множеству селекторов	`selector1, selector2`	Объединяет все элементы, отобранные каждым отдельным селектором, например: `input, select, textarea`

3.1.3. Селекторы псевдоклассов

Дополнительно библиотека jQuery поддерживает большое количество селекторов псевдоклассов, которые реализуют и дополняют псевдоклассы CSS. Псевдоклассы определяются стандартом CSS как классификаторы элементов, основанные на других характеристиках, кроме их имен, атрибутов или содержимого, которые невозможно определить из дерева документа. Эти селекторы начинаются с двоеточия (`:`), за которым следует имя селектора и необязательный параметр.

Например, чтобы найти все отмеченные элементы-флажки, можно воспользоваться комбинацией из двух селекторов псевдоклассов:

```
$('input:checkbox:checked')...
```

Имейте в виду, что большинство этих селекторов реализовано на языке JavaScript и их работа не может быть делегирована встроенным функциям браузеров. То есть они выполняются медленнее, чем простые селекторы.

Селекторы псевдоклассов делятся на множество различных типов. Обычные селекторы псевдоклассов исследуют только сами элементы – их атрибуты или содержимое, прежде чем принять решение о включении их в множество выбранных элементов. Некоторые селекторы псевдоклассов (*фильтры множеств*) принимают во внимание весь список элементов, полученный в результате предыдущей операции выбора, и выполняют фильтрацию, опираясь на позиции элементов в коллекции. Селекторы других типов (*фильтры потомков*) обращают внимание на отношения элементов с братскими узлами независимо от того, включены ли эти братские узлы в текущую коллекцию. При использовании селекторов их тип не имеет никакого значения для программиста – его приходится учитывать только при создании собственных селекторов.

В табл. 3.2 перечислены селекторы псевдоклассов, встроенные в библиотеку jQuery.

Таблица 3.2. Селекторы псевдоклассов jQuery

Селектор	Шаблон	Тип	Назначение
Селектор анимированных элементов	`:animated`		Выбирает все элементы, участвующие в воспроизведении анимационного эффекта, например: `div.content:animated`
Селектор кнопок	`:button`		Выбирает все элементы-кнопки и элементы `input` с атрибутом `type="button"`, например: `form :button`
Селектор флажков	`:checkbox`		Выбирает все элементы `input` с атрибутом `type="checkbox"`, например: `input:checkbox`
Селектор отмеченных элементов	`:checked`		Выбирает все отмеченные элементы (радиокнопки и флажки), например: `input[name="gender"]:checked`
Селектор по содержимому	`:contains (value)`		Выбирает все элементы, содержащие указанный текст (`value`), например: `h1:contains(Chapter)`
Селектор неактивных элементов	`:disabled`		Выбирает все неактивные элементы, например: `input:disabled`
Селектор пустых элементов	`:empty`		Выбирает все элементы, не имеющие дочерних узлов, в том числе и текстовых, например: `span:empty`
Селектор активных элементов	`:enabled`		Выбирает все активные элементы, например: `input:enabled`
Селектор по индексу	`:eq(index)`	Фильтр множеств	Выбирает элемент с указанным индексом (отсчет начинается с нуля) из предыдущего выбранного множества, например: `li:eq(1)`
Селектор четных элементов	`:even`	Фильтр множеств	Выбирает все элементы с четными индексами (отсчет начинается с нуля) из предыдущего выбранного множества, например: `tr:even`
Селектор файлов	`:file`		Выбирает все элементы `input` с атрибутом `type="file"`, например: `input:file`

Таблица 3.2 (продолжение)

Селектор	Шаблон	Тип	Назначение
Селектор первого элемента	`:first`	Фильтр множеств	Выбирает первый элемент из предыдущего выбранного множества, например: `select option:first`
Селектор первого потомка	`:first-child`	Фильтр потомков	Выбирает все элементы, которые являются первыми потомками своих родителей, например: `table td:first-child`
Селектор первого потомка заданного типа	`:first-of-type`	Фильтр потомков	Выбирает все элементы, которые являются первыми среди братских элементов с тем же именем (появился в jQuery 1.9), например: `p:first-of-type`
Селектор элемента с фокусом ввода	`:focus`		Выбирает элемент, который в настоящий момент владеет фокусом ввода, например: `input:focus`
Селектор элементов с большими индексами	`:gt(index)`	Фильтр множеств	Выбирает все элементы из предыдущего выбранного множества, которые имеют индексы больше указанного, например: `li:gt(1)`
Селектор по признаку наличия вложенных элементов	`:has(selector)`		Выбирает все элементы, содержащие хотя бы один элемент, соответствующий указанному селектору `selector`, например: `form:has(input.error)`
Селектор заголовков	`:header`		Выбирает все элементы-заголовки: `h1`, `h2` и т. д., например: `:header`
Селектор невидимых элементов	`:hidden`		Выбирает все невидимые элементы, например: `form input:hidden`
Селектор изображений	`:image`		Выбирает все элементы `input` с атрибутом `type="image"`, например: `input:image`
Селектор элементов ввода	`:input`		Выбирает все элементы ввода, включая раскрывающиеся списки, поля ввода текста и кнопки, например: `form :input`
Селектор языка	`:lang(language)`		Выбирает все элементы, для которых задан указанный язык `language` (появился в jQuery 1.9), например: `p:lang(fr)`

Таблица 3.2 (продолжение)

Селектор	Шаблон	Тип	Назначение
Селектор последнего элемента	`:last`	Фильтр множеств	Выбирает последний элемент из предыдущего выбранного множества, например: `#mylist li:last`
Селектор последнего потомка	`:last-child`	Фильтр потомков	Выбирает все элементы, которые являются последними потомками своих родителей, например: `table td:last-child`
Селектор последнего потомка заданного типа	`:last-of-type`	Фильтр потомков	Выбирает все элементы, которые являются последними среди братских элементов с тем же именем (появился в jQuery 1.9), например: `p:last-of-type`
Селектор элементов с меньшими индексами	`:lt(index)`	Фильтр множеств	Выбирает все элементы из предыдущего выбранного множества, которые имеют индексы меньше указанного, например: `li:lt(2)`
Селектор отрицания	`:not(selector)`		Выбирает все элементы, не соответствующие указанному селектору `selector`, например: `div:not(.ignore)`
Селектор *n*-го потомка	`:nth-child`	Фильтр потомков	Выбирает все элементы, которые являются *n*-ми потомками своих родителей, например: `table td:nth-child(3)`
Селектор *n*-го с конца потомка	`:nth-last-child`	Фильтр потомков	Выбирает все элементы, которые являются *n*-ми потомками своих родителей (в отличие от предыдущего селектора отсчет начинается с конца; появился в jQuery 1.9), например: `table td:nth-last-child(3)`
Селектор *n*-го с конца потомка заданного типа	`:nth-last-of-type(index)`	Фильтр потомков	Выбирает все элементы, которые являются *n*-ми среди братских элементов с тем же именем (отсчет начинается с конца; появился в jQuery 1.9), например: `p:nth-last-of-type(2)`
Селектор *n*-го потомка заданного типа	`:nth-of-type(index)`	Фильтр потомков	Выбирает все элементы, которые являются *n*-ми среди братских элементов с тем же именем (появился в jQuery 1.9), например: `p:nth-of-type(2)`

Таблица 3.2 (продолжение)

Селектор	Шаблон	Тип	Назначение
Селектор нечетных элементов	`:odd`	Фильтр множеств	Выбирает все элементы с нечетными индексами (отсчет начинается с нуля) из предыдущего выбранного множества, например: `tr:odd`
Селектор единственных потомков	`:only-child`	Фильтр потомков	Выбирает все элементы, являющиеся единственными потомками своих родителей, например: `li a:only-child`
Селектор единственных потомков заданного типа	`:only-of-type`	Фильтр потомков	Выбирает все элементы, не имеющие братских элементов с тем же именем (появился в jQuery 1.9), например: `p:only-of-type`
Селектор родителей	`:parent`		Выбирает все элементы, являющиеся родителями другого узла (включая текстовые), например: `li:parent`
Селектор полей ввода пароля	`:password`		Выбирает все элементы `input` с атрибутом `type="password"`, например: `input:password`
Селектор радиокнопок	`:radio`		Выбирает все элементы `input` с атрибутом `type="radio"`, например: `input:radio`
Селектор кнопок сброса	`:reset`		Выбирает все элементы `input` с атрибутом `type="reset"`, например: `form input:reset`
Селектор корня	`:root`		Выбирает элемент, являющийся корнем документа (появился в jQuery 1.9), например: `:root`
Селектор выбранных пунктов	`:selected`		Выбирает все выбранные элементы `select`, например: `#mylist :selected`
Селектор кнопок отправки форм	`:submit`		Выбирает все элементы `input` с атрибутом `type="submit"`, например: `input:submit`
Селектор фрагмента документа	`:target`		Выбирает элемент документа, на который ссылается идентификатор фрагмента в строке URI (появился в jQuery 1.9), например: `:target`

Таблица 3.2 (окончание)

Селектор	Шаблон	Тип	Назначение
Селектор текстовых полей ввода	`:text`		Выбирает все элементы `input` с атрибутом `type="text"` или `type="textarea"` например: `form :text`
Селектор видимых элементов	`:visible`		Выбирает все видимые элементы, например: `span:visible`

3.2. Добавление нового селектора псевдокласса

Добавление новых селекторов псевдоклассов производится через точку интеграции `$.expr.pseudos` (или `$.expr.filters` в версиях jQuery ниже 1.8.0). При этом достаточно всего лишь определить имя селектора и функцию, реализующую выбор. Создание селекторов других типов выглядит сложнее, так как библиотека Sizzle, лежащая в основе механизма выбора, не предусматривает возможности подобного расширения.

3.2.1. Структура селектора псевдокласса

Возьмем в качестве примера реализацию встроенного селектора `:has`, который возвращает элементы, содержащие хотя бы один элемент, соответствующий указанному селектору. Следующий селектор выбирает все элементы `fieldset`, содержащие элемент `input` с классом `error`:

```
$('fieldset:has(input.error)')...
```

> В механизм выбора Sizzle были внесены существенные изменения, которые отразились в версиях jQuery 1.8.0 и выше. Для полноты картины в этой главе мы рассмотрим способы добавления селекторов в версиях jQuery до и после 1.8.0.

Реализация селектора `:has`, как она определена в jQuery 1.8.0, приводится в листинге 3.1 (листинг 3.2 содержит реализацию из версии jQuery 1.7.2).

Листинг 3.1. Определение селектора :has (jQuery 1.8.0)

```
var Expr = Sizzle.selectors = {
    ...
    pseudos: {
        ...
        // ❶ Пометить как принимающий параметр
        "has": markFunction(function( selector ) {
```

```
            // ❷ Функция селектора
            return function( elem ) {
                // ❸ Принять или отвергнуть текущий элемент
                return Sizzle( selector, elem ).length > 0;
            };
        }),
        ...
    },
    ...
};
```

В версиях jQuery 1.8.0 и выше селекторы псевдоклассов имеют более простую организацию, чем в предыдущих версиях, и реализуются в виде функций, принимающих текущий элемент (`elem`) и возвращающих `true`, если селектор принимает элемент, и `false` в противном случае. Если для принятия решения селектору требуется параметр, этот параметр сохраняется в *замыкании* (в локальной области видимости, хранящей значения данных), окружающем заключительную функцию. Чтобы указать на необходимость параметра, функция замыкания помечается особым образом. Вам не придется беспокоиться об установке метки на функцию или заключать ее в замыкание, если селектор не имеет параметра.

Селектор `:has` имеет параметр – селектор для выбора вложенных элементов, поэтому функцию селектора необходимо заключить в замыкание, чтобы сохранить значение параметра, переданное пользователем (`selector`) ❶, и пометить ее (вызовом функции `markFunction` или ее псевдонима `createPseudo`), чтобы определить порядок вызова. Эта функция-обертка подключается к точке интеграции `Expr.pseudos`, которой позднее был присвоен псевдоним `$.expr.pseudos`. Функция селектора ❷ ищет элементы, соответствующие указанному селектору `selector`. Для этого она вызывает функцию поиска из библиотеки Sizzle, передавая ей текущий элемент в качестве контекста ❸. Исходя из количества найденных элементов, она возвращает `true` (если найден хотя бы один элемент) или `false`.

> Для обратной совместимости версии jQuery 1.8 и выше отображают прежде использовавшийся атрибут `filters` в новый атрибут `pseudos`. Благодаря этому старые расширения имеют возможность использовать прежнюю форму и работать со всеми версиями jQuery.

В листинге 3.2 представлена реализация селектора `:has`, как она определена в версии jQuery 1.7.2.

Листинг 3.2. Определение селектора :has (jQuery 1.7.2)

```
var Expr = Sizzle.selectors = {
    ...
```

```
    filters: {
        ...
        // ❶ Функция селектора
        has: function( elem, i, match ) {
            // ❷ Принять или отвергнуть текущий элемент
            return !!Sizzle( match[3], elem ).length;
    },
    ...
};
```

Функция селектора :has в библиотеке jQuery версии 1.7.2 ❶ принимает в качестве параметров текущий элемент (elem), его индекс в текущей коллекции (i) и массив фрагментов текста (match), выбранных регулярным выражением, которое было использовано механизмом Sizzle для извлечения селекторов. Полная строка селектора хранится в элементе массива с индексом 0 (match[0]), имя селектора – в элементе с индексом 1, кавычки (двойные или одинарные), окружающие значение в круглых скобках – в элементе с индексом 2, а параметры селектора – в элементе с индексом 3 (match[3]).

Функция селектора подключается к точке интеграции Expr.filters, на которую можно также сослаться по ее псевдониму $.expr.filters, и должна возвращать true, если селектор принимает указанный элемент, и false в противном случае. Достигается это за счет попытки найти соответствующий элемент в контексте текущего элемента ❷.

> Конструкция !! в языке JavaScript преобразует значение любого типа в значение типа Boolean. В JavaScript имеется несколько ложных значений, которые интерпретируются как значение false: false, NaN (Not a Number – не число), '' (пустая строка), 0 (число), null и undefined. Оператор ! преобразует любое значение в true или false и затем инвертирует результат, поэтому требуется второй оператор !, который восстановит значение типа Boolean в исходное состояние, соответствующее оригинальному значению.

Теперь, когда вы узнали, как реализуются селекторы псевдоклассов, можно попробовать создать свой селектор по точному соответствию содержимого, селектор по соответствию шаблону, селектор элементов списков или выделенных элементов, и селектор элементов, отмеченных как содержащие текст на иностранном языке.

3.2.2. Добавление селектора по точному соответствию содержимого

В библиотеке имеется встроенный селектор :contains, выбирающий элементы, которые где-то в своих недрах хранят текст, указанный

в параметре селектора. Но как быть, если потребуется отобрать только элементы, содержимое которых в точности совпадает с указанной строкой? Вы можете добавить свой селектор псевдокласса `:content`, выполняющий именно эту функцию.

Этот селектор, например, можно было бы использовать для поиска элементов списка, содержащих только слово «One»:

```
$('li:content(One)')...
```

В данном случае селектор должен извлечь все текстовое содержимое из тела элемента и сравнить его с указанным текстом, как показано в листингах 3.3 (для jQuery 1.8.0) и 3.4 (для jQuery 1.7.2).

Листинг 3.3. Селектор по точному соответствию содержимого (jQuery 1.8.0)

```
/* Извлекает полное текстовое содержимое элемента.
   @param element (элемент) элемент DOM, откуда извлекается содержимое
   @return (строка) текстовое содержимое элемента */
// ❶ Нормализует извлечение текста
function allText(element) {
    return element.textContent || element.innerText ||
           $.text([element]) || '';
}

/* Точное соответствие содержимого. */
// ❷ Определение селектора :content
$.expr.pseudos.content = $.expr.createPseudo(function(text) {
    // ❸ Принять или отвергнуть текущий элемент
    return function(element) {
        return allText(element) == text;
    };
});
```

Поскольку этот селектор требует передачи параметра, определяющего искомое содержимое, в версии jQuery 1.8.0 и выше необходимо создать замыкание для хранения значения параметра (`text`) и пометить эту функцию как требующую этого значения (вызовом `createPseudo`) ❷. Функция-обертка подключается к точке интеграции `$.expr.pseudos` с именем, соответствующим имени селектора. Сам селектор реализует внутренняя функция ❸. Она извлекает все текстовое содержимое из тела элемента (скрывая различия между браузерами за счет вызова функции `allText` ❶), сравнивает его с указанным текстом и возвращает `true`, чтобы принять текущий элемент, или `false`, чтобы отвергнуть его.

Листинг 3.4. Селектор по точному соответствию содержимого (jQuery 1.7.2)

```
/* Извлекает полное текстовое содержимое элемента.
   @param element (элемент) элемент DOM, откуда извлекается содержимое
   @return (строка) текстовое содержимое элемента */
// ❶ Нормализует извлечение текста
function allText(element) {
    return element.textContent || element.innerText ||
           $.text([element]) || '';
}

/* Точное соответствие содержимого. */
// ❷ Определение селектора :content
$.expr.filters.content = function(element, i, match) {
    return allText(element) == match[3];
};
```

Реализация для версии jQuery 1.7.2 выглядит похожей, но функция селектора принимает иные параметры и должна подключаться к точке интеграции `$.expr.filters` ❷. Функция `allText`, как и в предыдущем примере, извлекает текстовое содержимое из тела элемента ❶, которое затем сравнивается с заданным значением (`match[3]`), и по результатам сравнения текущий элемент принимается или отвергается возвратом значения `true` или `false`.

В настоящий момент этот код можно добавить только непосредственно перед обращением к функции обратного вызова `document.ready`, содержащей ваш код инициализации jQuery. В следующих главах вы увидите, как оформить свой код в виде самостоятельного плагина, которые легко можно было бы использовать в других страницах.

Селекторы для других версий jQuery

Для поддержки любых версий jQuery можно предварительно проверить наличие новых возможностей (функции `createPseudo`) и создать фильтр для соответствующей версии, как показано здесь. (Назначение оператора `!!` было описано в разделе 3.2.1.)

```
var usesCreatePseudo = !!$.expr.createPseudo; // jQuery 1.8+

if (usesCreatePseudo) {
    $.expr.pseudos.xxx = $.expr.createPseudo(function(param) {
        ...
    });
}
else {
    $.expr.filters.xxx = function(element, i, match) {
        ...
    };
}
```

3.2.3. Добавление селектора по соответствию шаблону

В предыдущем разделе было показано, как создать селектор по точному совпадению. Теперь вы готовы сделать еще шаг и реализовать селектор :matches для сопоставления содержимого элемента с регулярным выражением. Этот селектор можно было бы вызывать, как показано в табл. 3.3.

Таблица 3.3. Примеры использования селектора по соответствию шаблону

Селектор	Соответствует
`$('p:matches(One)')...`	Всем абзацам, содержащим текст «One»
`$('p:matches(One|Two)')...`	Всем абзацам, содержащим текст «One» или «Two»
`$('p:matches(~chapter \\d+)')...`	Всем абзацам, содержащим текст «chapter» (без учета регистра символов), за которым следует пробел и одна или две цифры
`$('p:matches("\\.\\.\\.$")')...`	Всем абзацам, заканчивающимся текстом «...» (каждую точку потребовалось экранировать, потому что в регулярных выражениях этот символ имеет специальное значение)

Как и в предыдущем примере, функция селектора извлекает текстовое содержимое элемента и сопоставляет его с указанным регулярным выражением. В листинге 3.5 приводится реализация для версии jQuery 1.8.0, а в листинге 3.6 – для версии jQuery 1.7.2. Обратите внимание, что регулярное выражение должно передаваться в виде строки, поэтому все встроенные обратные слэши необходимо экранировать.

Листинг 3.5. Селектор по соответствию шаблону (jQuery 1.8.0)

```
/* Сопоставляет содержимое элемента с регулярным выражением. */
// ❶ Определение селектора :matches
$.expr.pseudos.matches = $.expr.createPseudo(function(text) {
    // ❷ Фактическая функция селектора
    return function(element) {
        var flags = (text[0] || '') == '~' ? 'i' : '';
        // ❸ Принять или отвергнуть текущий элемент
        return new RegExp(text.substring(flags ? 1 : 0), flags).
            test(allText(element));
    };
});
```

Этот селектор также требует передачи параметра, в котором указывается шаблон для сопоставления. Поэтому в версии jQuery 1.8.0

и выше его необходимо завернуть в маркированную функцию (вызовом `createPseudo`), чтобы сохранить шаблон (`text`), и подключить к точке интеграции `$.expr.pseudos` ❶. Функция селектора ❷ создает регулярное выражение, опираясь на переданный шаблон, и сопоставляет с ним текстовое содержимое текущего элемента ❸, возвращая `true`, если соответствие найдено, и `false` в противном случае, чтобы принять или отвергнуть этот элемент.

К сожалению, селекторы псевдоклассов могут принимать только один аргумент. Если потребуется указать, что сопоставление должно выполняться без учета регистра символов, необходим какой-то способ сообщить об этом селектору, используя единственный параметр с шаблоном. Как показано выше, в табл. 3.3, в шаблон можно добавить начальный символ тильды (`~`) и интерпретировать его как требование выполнять сопоставление без учета регистра символов. Если потребуется добавить в начало шаблона литерал тильды, его можно экранировать символом обратного слэша (или двумя).

Листинг 3.6. Селектор по соответствию шаблону (jQuery 1.7.2)

```
/* Сопоставляет содержимое элемента с регулярным выражением. */
// ❶ Определение селектора :matches
$.expr.filters.matches = function(element, i, match) {
    var flags = (match[3][0] || '') == '~' ? 'i' : '';
    // ❷ Принять или отвергнуть текущий элемент
    return new RegExp(match[3].substring(flags ? 1 : 0), flags).
        test(allText(element));
};
```

В версии jQuery 1.7.2 функция селектора подключается к точке интеграции `$.expr.filters` и принимает несколько параметров: текущий элемент (`element`), индекс элемента в коллекции (`i`) и массив совпадений с шаблоном (`match`) ❶. Как и в более новой версии библиотеки, функция селектора создает объект регулярного выражения и выполняет сопоставление указанного шаблона с текстовым содержимым ткущего элемента ❷, возвращая `true`, чтобы принять элемент, или `false` в противном случае.

Регулярные выражения в JavaScript

Применение регулярных выражений в JavaScript играет важную роль в эффективном использовании языка. Регулярные выражения часто будут встречаться в этой книге в примерах расширений, где они используются для проверки значений или для разбиения строк на составляющие их компоненты. Вам обязательно следует познакомиться с их синтаксисом и шаблонами использования. Общие сведения об использовании регулярных выражений вы найдете в приложении в конце книги. Более подробную

информацию можно найти в различных справочниках и руководствах, доступных в Интернете:

- «JavaScript RegExp Object» – www.w3schools.com/jsref/jsref_obj_regexp.asp[1];
- «Регулярные выражения» – https://developer.mozilla.org/ru/JavaScript/Guide/Regular_Expressions;
- «Использование регулярных выражений» – www.regular-expressions.info/javascript.html[2];
- «Regular Expression Tutorial» – www.learn-javascript-tutorial.com/RegularExpressions.cfm[3].

3.2.4. Добавление селектора по типу элемента

Селекторы псевдоклассов можно также использовать для упрощения отбора элементов нескольких типов по аналогии со встроенным селектором `:input`, которому соответствуют элементы `input`, `select`, `textarea` и `button`.

Например, можно написать селектор `:list`, отбирающий элементы нумерованных и маркированных списков, или селектор `:emphasis`, отбирающий выделенные элементы, и использовать их, как показано ниже:

- `$('#main :list')...`, чтобы выбрать все нумерованные и маркированные списки в элементе `#main`;
- `$('p:has(:emphasis)')...`, чтобы выбрать все абзацы, содержащие выделенные элементы.

Оба селектора используют простые регулярные выражения для сопоставления с именем узла элемента, как показано в листинге 3.7. Так как эти селекторы не принимают параметров, они не требуют специальной обработки в jQuery 1.8.0, благодаря чему в обеих версиях библиотеки может использоваться один и тот же код реализации.

Листинг 3.7. Селекторы списков и разметки выделения текста

```
/* Все списки. */
// ❶ Селектор :list
$.expr.filters.list = function(element) {
    return /^(ol|ul)$/i.test(element.nodeName);
};
```

---

[1] Похожая статья на русском языке: http://javascript.ru/RegExp. – *Прим. перев.*

[2] Похожая статья на русском языке: http://www.wisdomweb.ru/JS/obreg.php. – *Прим. перев.*

[3] Похожие руководства на русском языке: http://javascript.ru/basic/regular-expression и http://htmlweb.ru/java/regexp.php – *Прим. перев.*

```
/* Выделенный текст. */
// ❷ Селектор :emphasis
$.expr.filters.emphasis = function(element) {
    return /^(b|em|i|strong)$/i.test(element.nodeName);
};
```

Так как точка интеграции `$.expr.filters` определена в jQuery 1.8.x как псевдоним для `$.expr.pseudos`, ее можно использовать во всех версиях jQuery. Функция селектора элементов списков подключается к этой точке интеграции и с помощью регулярного выражения проверяет совпадение имени элемента с `ol` или `ul` ❶. Это выражение ищет совпадение с одной из заданных альтернатив (`|`) с начала (`^`) и до конца (`$`) текста. Аналогично функция селектора выделенного текста сопоставляет имя элемента с альтернативами `b`, `em`, `i` и `strong` ❷, опять же с помощью регулярного выражения.

3.2.5. Добавление селектора элементов с текстом на иностранном языке

Селекторы могут проверять не только имена элементов или их содержимое – для исследования доступно все, что имеет отношение к элементу. Например, можно создать селектор псевдокласса, выбирающий элементы с содержимым на иностранном языке.

Этот селектор можно было бы использовать, как показано ниже:

- `$('p:foreign')...`, чтобы выбрать все абзацы с содержимым на языке, отличном от языка по умолчанию;
- `$('p:foreign(fr)')...`, чтобы выбрать все абзацы, отмеченные как содержащие текст на французском языке.

Селектор ищет атрибут `lang` в элементах и действует в одном из двух режимов. При использовании селектора без параметра он выбирает все элементы с атрибутом `lang`, кроме тех, значение которых совпадает с текущим языком браузера. Если селектору передан определенный язык, он возвращает только элементы, помеченные этим языком. В листинге 3.8 приводится код селектора для jQuery 1.8.0, а в листинге 3.9 – для jQuery 1.7.2.

> Селектор `:lang`, добавленный в версии jQuery 1.9.0, предоставляет аналогичную функциональность.

Листинг 3.8. Селектор элементов с текстом на иностранном языке (jQuery 1.8.0)

```
/* Язык браузера по умолчанию. */
// ❶ Создать функцию проверки языка браузера по умолчанию
```

```
var defaultLanguage = new RegExp('^' +
    (navigator.language || navigator.userLanguage).substring(0, 2), 'i');

/* Элементы с текстом на иностранном языке. */
// ❷ Определение селектора :foreign
$.expr.pseudos.foreign = $.expr.createPseudo(function(language) {
    return function(element) {
        // ❸ Извлечь определение языка из элемента
        var lang = $(element).attr('lang');
        return !!lang && (!language ? !defaultLanguage.test(lang) :
            // ❹ И сравнить
            new RegExp('^' + language.substring(0, 2), 'i').
                test(lang));
    };
});
```

Здесь сначала создается регулярное выражение, проверяющее совпадение с языком браузера по умолчанию ❶. Данный селектор может принимать параметр, поэтому в версиях jQuery 1.8.0 и выше для его определения необходимо использовать помеченную функцию-обертку (`createPseudo`), чтобы сохранить значение параметра (`language`) ❷, а для подключения должна использоваться точка интеграции `$.expr.pseudos`. Затем функция селектора отыскивает атрибут `lang` в текущем элементе ❸ и выполняет проверку в одном из двух режимов, в зависимости от наличия параметра. Если селектор вызван без параметра, он выбирает все элементы с атрибутом `lang`, значение которого не совпадает с текущим языком браузера ❹. Если селектору передан какой-то определенный язык, он возвращает `true` только для элементов, отмеченных этим языком.

Описание конструкции `!!` приводится в разделе 3.2.1.

Листинг 3.9. Селектор элементов с текстом на иностранном языке (jQuery 1.7.2)

```
/* Язык браузера по умолчанию. */
var defaultLanguage = new RegExp('^' +
    (navigator.language || navigator.userLanguage).substring(0, 2), 'i');

/* Элементы с текстом на иностранном языке. */
// ❶ Определение селектора :foreign
$.expr.filters.foreign = function(element, i, match) {
    // ❷ Извлечь определение языка и сравнить
    var lang = $(element).attr('lang');
    return !!lang && (!match[3] ? !defaultLanguage.test(lang) :
        new RegExp('^' + match[3].substring(0, 2), 'i').test(lang));
```

Код для jQuery 1.7.2 почти не изменился, по сравнению с кодом для более новой версии библиотеки. Разница лишь в том, что функция селектора принимает все параметры непосредственно и подключается к точке интеграции `$.expr.filters` ❶. Как и в предыдущем примере, она извлекает атрибут `lang` из текущего элемента и сопоставляет его со значением параметра (`match[3]`) ❷.

3.2.6. Селекторы из расширения Validation

Селекторы, представленные в листинге 3.10, определены в расширении Validation, написанном Йерном Заффererом (Jörn Zaefferer)[1]. Эти селекторы позволяют находить все поля без значений, поля с определенным значением или все неотмеченные элементы. Так как они не принимают параметров, селекторы имеют идентичную реализацию для всех версий jQuery.

Листинг 3.10. Селекторы из расширения Validation

```
// Собственные селекторы
// ❶ Добавить селекторы псевдоклассов
$.extend($.expr[":"], {
    // ❷ Соответствует пустым полям
    // http://docs.jquery.com/Plugins/Validation/blank
    blank: function(a) {return !$.trim("" + a.value);},
    // ❸ Соответствует непустым полям
    // http://docs.jquery.com/Plugins/Validation/filled
    filled: function(a) {return !!$.trim("" + a.value);},
    // ❹ Соответствует неотмеченным элементам
    // http://docs.jquery.com/Plugins/Validation/unchecked
    unchecked: function(a) {return !a.checked;}
});
```

Вместо точки интеграции `$.expr.filters` Йерн использует точку интеграции `$.expr[":"]` ❶, которая фактически является псевдонимом для `$.expr.filters`. Функция селектора `blank` возвращает `true`, если поле не имеет значения (включая пробелы) ❷. (Напомню, что пустые строки интерпретируются в JavaScript как значение `false`.) Функция селектора `filled`, напротив, возвращает `true`, если поле имеет некоторое значение ❸ (инвертирует результат, возвращаемый предыдущей функцией `blank`). Функция селектора `unchecked` возвращает `true`, если атрибут `checked` поля имеет значение `null` или `false` ❹.

Обратите внимание, что эти селекторы возвращают множество элементов, поэтому их можно объединять с другими селекторами элементов форм, например `:checkbox:unchecked`.

---

[1] http://jqueryvalidation.org/.

3.3. Добавление фильтров множеств

Другой тип селекторов псевдоклассов – *фильтры множеств* (set filters). Если селекторы, рассматривавшиеся выше, анализировали единственный узел, фильтры множеств получают целую коллекцию элементов. Типичными примерами фильтров множеств являются селекторы псевдоклассов :first, :last, :odd и :even.

> В механизм выбора Sizzle были внесены существенные изменения, которые отразились в версиях jQuery 1.8.0 и выше. Для полноты картины в этой главе мы рассмотрим способы добавления селекторов в версиях jQuery до и после 1.8.0.

3.3.1. Структура селектора множества

Селектор :last – это встроенный фильтр множеств, возвращающий последний элемент прежде выбранной коллекции элементов. Давайте возьмем этот селектор в качестве примера и посмотрим, как действуют фильтры множеств. В листинге 3.11 приводится реализация селектора для jQuery 1.8.0, а в листинге 3.12 – для jQuery 1.7.2.

Листинг 3.11. Определение селектора :last (jQuery 1.8.0)

```
var Expr = Sizzle.selectors = {
    ...
    setFilters: {
        ...
        // ❶ Определение селектора :last
        "last": function( elements, argument, not ) {
            var elem = elements.pop();
            // ❷ Вернуть отфильтрованное множество
            return not ? elements : [ elem ];
        },
        ...
    }
};
```

В версии jQuery 1.8.0 и выше новые фильтры множеств подключаются к точке интеграции Expr.setFilters (или к ее псевдониму $.expr.setFilters). При этом указываются имя нового фильтра и функция, обрабатывающая текущее множество элементов ❶. В качестве параметров функции передаются текущий массив элементов (elements), параметр, указанный при вызове селектора (argument), и логический флаг, сообщающий о необходимости инвертировать действие фильтра (not). Функция должна возвращать отфильтрованное множество элементов в виде массива ❷. jQuery 1.8.0 выполняет единственный вызов функции фильтра множества, которая должна обработать сразу весь список и вернуть выбранные элементы с учетом значения not.

В jQuery 1.8.2 были внесены дополнительные изменения в функцию setFilters. Теперь внутри jQuery вызывает функцию createPositionalPseudo для фильтров. Но эта функция недоступна извне, и новая реализация setFilters все еще может использоваться, как показано выше.

Листинг 3.12. Определение селектора :last (jQuery 1.7.2)

```
var Expr = Sizzle.selectors = {
    ...
    setFilters: {
        ...
        // ❶ Определение селектора :last
        last: function( elem, i, match, array ) {
            // ❷ Принять или отвергнуть текущий элемент
            return i === array.length - 1;
        },
        ...
    },
    ...
};
```

В версиях, предшествовавших версии jQuery 1.8.0, функция селектора также подключается к точке интеграции Expr.setFilters (или ее псевдониму $.expr.setFilters), но вызывается для каждого элемента в отдельности ❶. В виде параметров функции передаются: текущий элемент (elem), его позиция в списке (i), компоненты совпадения с регулярным выражением, выбирающим селектор (match), и коллекция элементов, составляющих список (array). Функция должна вернуть true, если текущий элемент принимается, или false, если отвергается ❷.

Идентификация фильтров множеств до версии jQuery 1.8.2

Так как в версиях, предшествовавших версии jQuery 1.8.2, фильтры множеств интерпретируются иначе, чем другие селекторы псевдоклассов, они должны идентифицироваться некоторым способом, чтобы обеспечить их корректный вызов библиотекой. В jQuery это достигается явным сопоставлением их имен внутри механизма Sizzle с регулярным выражением, извлекающим их из строки выбора $.expr.match.POS, для позиционных селекторов.

```
var pos = ":(nth|eq|gt|lt|first|last|even|odd)(?:\\((\\d*)\\)|)(?=[^-]|$)";
$.expr.match.POS = new RegExp( pos, "ig" );
```

Чтобы добавить новый фильтр множеств в ранние версии jQuery, необходимо так же добавить его имя в этот список. Если этого не сделать, при попытке использовать селектор вы получите сообщение об ошибке: «Syntax error, unrecognized expression: unsupported pseudo: xxxxxx» (Синтаксическая ошибка, неопознанное выражение: неподдерживаемый псевдокласс: xxxxxx). О том, как добавлять имена при создании новых селекторов, рассказывается в следующем разделе.

Версия jQuery 1.8.2 идентифицирует фильтры множеств автоматически и не требует вручную обновлять регулярное выражение `$.expr.match.POS`.

Используя приемы, описанные выше, можно добавлять собственные фильтры множеств, например, для выборки элементов в середине списка или для выборки элементов по индексам, откладываемым с конца коллекции.

3.3.2. Добавление селектора выборки элементов из середины множества

Для обработки списков элементов можно определить селектор `:middle`, отбрасывающий первый и последний элементы множества. Этот селектор можно было бы использовать, как показано ниже:

```
$('li:middle')...
```

В листинге 3.13 представлена реализация селектора для jQuery 1.8.0, а в листинге 3.14 – для jQuery 1.7.2.

Листинг 3.13. Селектор выборки элементов из середины коллекции (jQuery 1.8.0)

```
// ❶ Для опознания нового селектора
$.expr.match.POS = new RegExp(
    $.expr.match.POS.source.replace(/odd/, 'odd|middle'), 'ig');

/* Элементы в середине. */
// ❷ Определение селектора :middle
$.expr.setFilters.middle = function(elements, argument, not) {
    // ❸ Вернуть отфильтрованное множество
    var firstLast = [elements.shift(), elements.pop()];
    return not ? firstLast : elements;
};
```

В версиях jQuery 1.8.0 и 1.8.1 необходимо добавить имя селектора в регулярное выражение, извлекающее имена селекторов из полной строки выбора, поэтому в листинге 3.13 выполняется переопределение регулярного выражения `$.expr.match.POS`, как описывалось в предыдущем разделе, чтобы вставить имя нового селектора в конец списка существующих селекторов ❶. Определение существующего регулярного выражения извлекается из атрибута `source`, затем выполняются замена текста `odd` на `odd|middle` и перекомпиляция новой версии выражения. Не забудьте включить флаги `ig`, обеспечивающие глобальный поиск совпадений (`g`) без учета регистра символов (`i`). Изменять регулярное выражение в jQuery 1.8.2 не требуется.

Начиная с версии jQuery 1.8.0, селектор должен подключаться к точке интеграции `$.expr.setFilters`, обрабатывать коллекцию элементов целиком и возвращать отфильтрованный список ❷. В данном случае из списка удаляются первый и последний элементы (с применением стандартных функций `shift` и `pop`) и сохраняются в новом массиве ❸. Исходя из значения параметра `not`, возвращается либо новый массив с двумя элементами (если в параметре `not` получено значение `true`), либо оставшаяся часть массива (если в параметре `not` получено значение `false`).

Листинг 3.14. Селектор выборки элементов из середины коллекции (jQuery 1.7.2)

```
// ❶ Для опознания нового селектора
$.expr.match.POS = new RegExp(
    $.expr.match.POS.source.replace(/odd/, 'odd|middle'));
$.expr.leftMatch.POS = new RegExp(
    $.expr.leftMatch.POS.source.replace(/odd/, 'odd|middle'));

/* Элементы в середине. */
// ❷ Определение селектора :middle
$.expr.setFilters.middle = function(element, i, match, list) {
    // ❸ Принять или отвергнуть текущий элемент
    return i > 0 && i < list.length - 1;
};
```

В ранних версиях jQuery необходимо также было добавить имя нового фильтра множеств в регулярное выражение, извлекающее их из общей строки выбора. В этом примере снова имя селектора добавляется в регулярное выражение `$.expr.match.POS` ❶. В отличие от предыдущего примера, в ранних версиях jQuery не требуется добавлять флаги `ig`, но требуется также добавить новое имя во второе регулярное выражение, порожденное из первого (в jQuery 1.8.0 эта операция выполняется автоматически). Если вы пользуетесь одной из ранних версий jQuery, вам следует обновить `$.expr.leftMatch.POS` таким же способом, как и `$.expr.match.POS`.

Функция селектора здесь обрабатывает каждый элемент множества в отдельности ❷. В данном случае она сравнивает позицию элемента внутри множества и отвергает первый и последний элементы, возвращая `false` ❸.

3.3.3. Расширение селектора по индексу

Один из недостатков стандартной реализации селектора `:eq` в библиотеке jQuery – отсутствие поддержки отрицательных индексов, опре-

деляющих позиции элементов от конца коллекции даже при том, что эта возможность предусмотрена в соответствующей функции `eq()`.

Мы легко можем добавить эту поддержку всего двумя строчкам кода. В результате мы сможем использовать селектор, как показано ниже:

- `$('li:eq(1)')...`, чтобы выбрать второй элемент списка;
- `$('li:eq(-2)')...`, чтобы выбрать предпоследний элемент списка.

Начиная с версии jQuery 1.8.1, эта функциональность стала стандартной и не требует добавления в виде расширения.

Однако в данный момент передача знака «минус» в параметре селектора `:eq` приводит к тому, что выражение игнорируется и селектор ничего не возвращает или генерирует ошибку. Для начала нам необходимо обеспечить возможность передачи знака «минус», прежде чем добавлять новую функциональность.

В листинге 3.15 приводится код для 1.8.0, а в листинге 3.16 – для jQuery 1.7.2.

Листинг 3.15. Расширение селектора по индексу (jQuery 1.8.0)

```
// ❶ Разрешить прием отрицательных значений в параметре
$.expr.match.POS = new RegExp(
    $.expr.match.POS.source.replace(/\\d\*/, '-?\\d*'), 'ig');

/* Добавить поддержку отрицательных индексов. */
// ❷ Переопределить селектор :eq
$.expr.setFilters.eq = function(elements, argument, not) {
    // ❸ Вычислить позицию
    argument = parseInt(argument, 10);
    argument = (argument < 0 ? elements.length + argument : argument);
    // ❹ Вернуть отфильтрованное множество элементов
    var element = elements.splice(argument, 1);
    return not ? elements : element;
};
```

Поддержка отрицательных индексов обеспечивается корректировкой регулярного выражения, используемого фильтрами множеств: шаблон `$.expr.match.POS` в механизме выбора Sizzle. Здесь мы переопределили выражение так, чтобы оно распознавало необязательный знак «минус» перед числами внутри селектора ❶. Внутри текущего регулярного выражения `POS`, которое можно извлечь из атрибута `source`, отыскивается подвыражение `\d*`, соответствующее последовательности из любого числа цифр, и замещается подвыражением `-?\d*`, добавляющим поддержку необязательного знака «минус». Обратите

внимание на необходимость экранирования обратных слэшей, чтобы они интерпретировались как литералы.

Затем переопределяется сама функция `eq`, добавляющая поддержку отрицательных индексов ❷. Если она обнаруживает в параметре отрицательное значение, это значение складывается со значением длины списка элементов (не забывайте, что число отрицательное, и поэтому операция сложения фактически превращается в операцию вычитания его абсолютного значения) и полученный результат используется как индекс для сравнения ❸. Положительные индексы обрабатываются, как и прежде. Далее элемент с вычисленным индексом удаляется из списка, и вызывающей программе возвращается этот элемент или остаток списка, в зависимости от значения параметра `not` ❹.

Листинг 3.16. Расширение селектора по индексу (jQuery 1.7.2)

```
// ❶ Разрешить прием отрицательных значений в параметре
$.expr.match.POS = new RegExp(
    $.expr.match.POS.source.replace(/\\d\*/, '-?\\d*'));
$.expr.leftMatch.POS = new RegExp(
    $.expr.leftMatch.POS.source.replace(/\\d\*/, '-?\\d*'));

/* Добавить поддержку отрицательных индексов. */
// ❷ Переопределить селектор :eq
$.expr.setFilters.eq = function(element, i, match, list) {
    // ❸ Вычислить позицию
    var index = parseInt(match[3], 10);
    index = (index < 0 ? list.length + index : index);
    // ❹ Принять или отвергуть текущий элемент
    return index === i;
};
```

Реализация для jQuery 1.7.2 близко напоминает реализацию для 1.8.0. Помимо обычного шаблона фильтра (`$.expr.match.POS`) необходимо так же подобным образом изменить соответствующий шаблон `$.expr.leftMatch.POS` ❶, как это делалось в примере селектора `:middle`. Затем переопределяется функция `eq` ❷, которая извлекает значение параметра из компонента (`match[3]`) и вычисляет требуемый индекс, опираясь на знак параметра ❸. Функция возвращает `true`, чтобы принять текущий элемент, если его позиция совпадает с требуемой, или `false` в противном случае ❹.

Это нужно знать

Селекторы псевдоклассов следует создавать для стандартизации или увеличения очевидности процедуры выбора.

Для подключения селекторов псевдоклассов служит точка интеграции `$.expr.pseudos` (`$.expr.filters` – в версиях jQuery, предшествовавших версии 1.8).
Для сохранения значения параметра в замыкании следует использовать функцию `$.expr.createPseudo`.
Для подключения фильтров множеств служит точка интеграции `$.expr.setFilters`.
Регулярные выражения JavaScript могут помочь в сопоставлении шаблонов символов.

Попробуйте сами

Создайте замену стандартному селектору `:header`, который в дополнение к стандартным элементам (`h1–h6`) извлекал бы новый элемент `header`, появившийся в HTML5.

3.4. В заключение

Работа библиотеки jQuery основана на процедуре выбора коллекций элементов, над которыми затем выполняются некоторые операции. Она предлагает несколько способов доступа к элементам – по имени элемента, по значению атрибута `id` или `class`, по значению любого другого атрибута, а также с применением селекторов псевдоклассов – и дает возможность объединять их в строке запроса. Несмотря на богатство возможностей, иногда проще бывает определить собственный селектор и использовать его, чтобы более кратко и ясно выражать свои намерения.

В этой главе вы увидели, как создаются новые, простые селекторы псевдоклассов, такие как `:matches` и `:emphasis`, которые исследуют узлы по отдельности. Затем вы узнали, как определять фильтры множеств, такие как `:middle`, которые анализируют текущую коллекцию элементов целиком. Используя описанные в данной главе приемы, можно создавать собственные селекторы и тем самым упрощать процесс разработки.

В следующих главах вы познакомитесь с некоторыми принципами проектирования расширений для jQuery и инфраструктурой, реализующей эти принципы. Вы также узнаете, как упаковывать свои расширения в автономные модули, которые можно повторно использовать в других страницах.

Часть II

Расширения и функции

Чаще всего сторонние разработчики создают расширения, которые оперируют коллекциями элементов, выбранных из документа. В этой части книги рассматриваются наиболее удачные приемы создания таких расширений.

В главе 4 рассматривается ряд основных принципов создания расширений, которых следует придерживаться при создании собственных расширений. Следование этим принципам поможет вам создавать надежные и практичные расширения.

В главе 5 мы пройдем от начала до конца всю процедуру создания расширения коллекций, то есть расширения, выполняющего операции с коллекцией элементов страницы. Использование инфраструктуры поддержки расширений поможет применить принципы, описываемые в главе 4, и при этом все свое внимание сосредоточить на реализации фактической функциональности расширения.

Расширения-функции не предназначены для работы с выбранными элементами. Их цель – расширить функциональные возможности страницы в целом непротиворечивым образом. В главе 6 исследуются два примера расширений такого рода, помогающие решать проблемы локализации и использования cookies.

Чтобы привлечь внимание как можно более широкого круга пользователей к своему расширению, его необходимо тщательно протестировать и упаковать, описать в документации его возможности и предоставить демонстрационные примеры его применения. Все эти аспекты разработки рассматриваются в главе 7.

Глава 4

Принципы разработки расширений

Эта глава охватывает следующие темы:

- архитектура расширений;
- руководящие принципы разработки.

Расширения для jQuery имеют широчайшую область применения – от простого изменения классов элементов и обработки событий до реализации селекторов, анимационных эффектов, полнофункциональных графических виджетов и средств удаленного доступа. Возможности расширений ограничиваются только вашей фантазией.

В предыдущей главе вы узнали, как создавать собственные селекторы для использования совместно с библиотекой jQuery. Вашему вниманию было представлено несколько простых примеров одного из способов расширения ее возможностей. Теперь мы отступим на шаг назад и окинем одним взглядом архитектуру расширений и процесс их создания. Независимо от типа создаваемого расширения вы должны придерживаться описываемых далее принципов, чтобы обеспечить жизнеспособность своего расширения и широкий интерес к нему в экосистеме jQuery и JavaScript.

В этой главе не только обсуждаются упомянутые принципы, но и разъясняется, почему следует их придерживаться. Здесь также рассматриваются преимущества оформления расширений в виде модулей (плагинов) перед их встраиванием в веб-страницы и некоторые вопросы проектирования расширений.

4.1. Архитектура расширений

Первое, с чем следует определиться, – какие функциональные возможности должны поддерживаться расширением. Обычно такое решение принимается, исходя из конкретных потребностей ваших про-

ектов. Если вы обнаруживаете, что снова и снова добавляете в свои веб-страницы одни и те же функциональные возможности, это может служить поводом задуматься о создании собственного расширения, которое можно было бы использовать по мере необходимости. Например, это может быть несколько виджетов раскрывающихся списков, взаимодействующих друг с другом, которые позволяют выбирать дату и гарантируют, что доступное число месяца не превысит количество дней в выбранном месяце.

4.1.1. Преимущества оформления расширений в виде модулей

Оформление расширений в виде модулей (плагинов) дает следующие преимущества:

- простота повторного использования;
- непротиворечивость;
- простота сопровождения.

Оформляйте свои расширения в виде модулей, чтобы максимально упростить повторное их использование. При таком подходе вы сможете всего парой строк кода загрузить модуль в веб-страницу и применить его возможности к выбранным элементам, как показано в листинге 4.1. Обеспечьте возможность использования своего кода в схожих ситуациях, предоставляя параметры настройки поведения расширения.

Листинг 4.1. Загрузка и вызов модуля

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/jquery.myplugin.js"></script>
<script type="text/javascript">
$(function() {
    $('.myelements').myplugin({option1: true, option2: 'XYZ'});
});
</script>
```

Благодаря поддержке стилей оформления и многократного использования вы сможете обеспечить доступность одних и тех же функциональных возможностей и единообразие визуального оформления всех своих проектов.

Если в коде обнаружится какая-либо ошибка, достаточно будет внести изменения в одном месте, чтобы исправить проблему во всех своих проектах. Всестороннее тестирование также потребуется выполнить только один раз, поскольку во всех проектах будет использоваться одна и та же реализация расширения. Тесты должны быть

как можно более простыми, чтобы их проще было повторять в разных окружениях. Аналогично любые усовершенствования в расширении немедленно будут доступны во всем приложении.

4.1.2. Проектирование архитектуры

Определившись с функциональными возможностями расширения, можно приступать к проектированию его архитектуры и рассмотрению вопросов, перечисленных ниже.

- Как расширение должно выглядеть на странице (если речь идет о графическом виджете)?
- Как расширение будет взаимодействовать с пользователем – посредством клавиатуры, мыши или программно?
- Какую информацию должен хранить каждый экземпляр? Какая информация должна быть общей для всех экземпляров?
- Как пользователь будет настраивать внешний вид или поведение расширения?
- Какие внутренние события будут представлять интерес для пользователя?

Не будьте слишком амбициозны. Сосредоточьтесь на решении одной проблемы. С течением времени у вас часто будет возникать соблазн добавить в расширение новые возможности. Но, прежде чем приступить к воплощению новых идей, задайте себе следующие вопросы.

- Насколько тесно связаны новые возможности с главной целью расширения?
- Насколько востребованы эти возможности? Не получится ли так, что в большинстве случаев они окажутся избыточными?
- Можно ли сделать новые возможности как можно более универсальными, чтобы максимально расширить круг их применения?

Например, ранняя версия моего расширения Datepicker поддерживала возможность отображения в виде диалога. На тот момент такая возможность казалась достаточно практичной, так как позволяла получить дату от пользователя. Но она вносила излишнюю сложность в код и снижала его гибкость. Со временем обнаружилось, что расширение Datepicker легко можно добавить в диалог, выводимый расширением Dialog, которое к тому же дает возможность встраивать дополнительное содержимое в окно диалога. Теперь поддержка вывода диалога обладает дополнительной функциональностью, такой как возможность буксировки и изменения размеров окна мышью, без необходимости дублировать существующий код, и расширение Date-

picker стало менее сложным. Для сравнения на рис. 4.1 изображены две версии расширения.

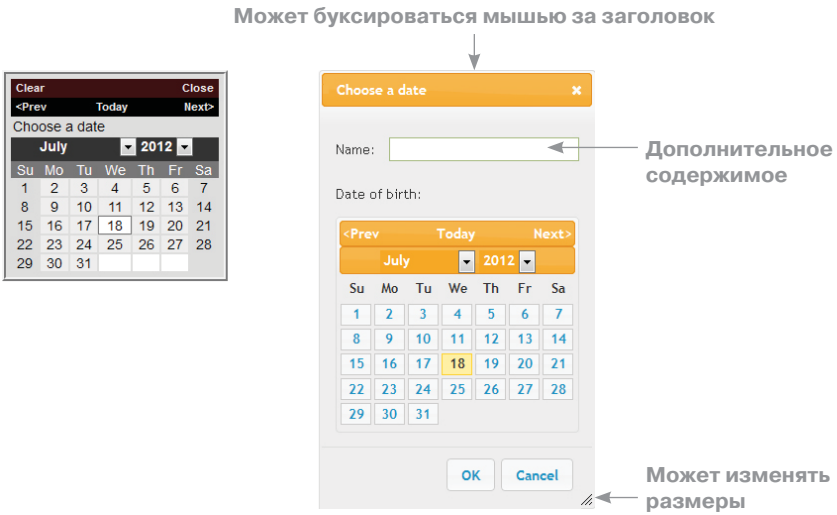


Рис. 4.1 ❖ Встроенный диалог в прежней версии Datepicker (слева) и новый диалог из библиотеки jQuery UI со встроенным виджетом Datepicker (справа)

4.1.3. Поддержка модульной архитектуры в расширениях

Если новая функциональность оказалась достаточно сложной, чтобы возникло желание избежать повторения кода, но слишком узкоспециализированной, чтобы быть широко востребованной, ее можно выделить в отдельный модуль и загружать, только когда это действительно необходимо. Благодаря такому подходу вы сохраняете все преимущества представления расширений в виде отдельных модулей, даете пользователям возможность загружать только базовую функциональность, кому этого достаточно, и подключать расширенные возможности тем, кто действительно нуждается в них.

Например, расширение вывода графиков может иметь несколько дополнительных модулей для отображения редко используемых диаграмм, таких как диаграммы рассеивания, полярные диаграммы и картосхемы. Большинство пользователей будут пользоваться выгодами, которые несут уменьшение объема загружаемого кода, но те,

кому подобные диаграммы необходимы, смогут подключать их одной строкой кода.

4.2. Руководящие принципы

Ниже я познакомлю вас с основными принципами, которым стараюсь следовать при создании собственных расширений. Они представляют наиболее удачные, в моем понимании, приемы проектирования расширений. Следование этим принципам поможет вам минимизировать конфликты вашего расширения с обширной экосистемой jQuery и JavaScript и одновременно защитят ваш код от влияния извне.

> Большая часть этих принципов реализована в инфраструктуре поддержки расширений, которая описывается в главе 5, а также в инфраструктуре поддержки виджетов для jQuery UI, о которой рассказывается в главе 8. Обсуждение этих принципов и их реализацию я оставляю до указанных глав.

4.2.1. Наращивайте возможности прогрессивно

В идеале расширения должны обеспечивать прогрессивное наращивание функциональных возможностей веб-страниц. Это означает, что страницы должны сохранять свою работоспособность, даже если пользователь отключил поддержку JavaScript в своем браузере, а пользователи с включенной поддержкой должны получать более удобный интерфейс и расширенные возможности.

Расширение Datepicker является отличным примером следования этому принципу. В браузере без поддержки JavaScript пользователь получит текстовое поле ввода, куда сможет ввести дату, надеясь, что формат ее представления был выбран правильно. При наличии поддержки JavaScript он получит возможность открыть календарь и выбрать желаемую дату мышью, как показано на рис. 4.2. Календарь обеспечивает более наглядный интерфейс выбора даты, способен ограничивать диапазон доступных дат и предлагает дополнительные средства визуального выделения некоторых дней. Выбор даты в календаре приводит к заполнению поля ввода, как если бы дата была введена вручную, но гарантирует, что она будет иметь допустимый формат.

4.2.2. Объявляйте только одно имя и используйте его повсюду

Добавляйте в пространство имен jQuery не более одного имени и используйте его повсюду для ссылки на свое расширение. Следование

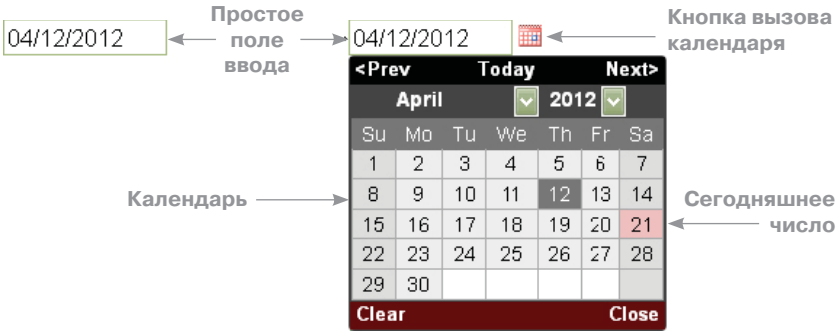


Рис. 4.2 ❖ Расширение Datepicker прогрессивно наращивает возможности стандартного поля ввода

этому принципу поможет минимизировать конфликты имен между вашими и другими расширениями.

Имя расширения должно быть достаточно длинным, чтобы уменьшить вероятность конфликтов имен, и отражать назначение расширения. Хорошими примерами могут служить имена `datepicker` и `validate`. К сожалению, имя `slider` в этом отношении недостаточно хорошо. Оно используется внутри библиотеки jQuery UI для идентификации элемента управления, позволяющего выбрать значение внутри определенного диапазона, а также многими другими расширениями, реализующими управление последовательностями изображений или фрагментов содержимого и осуществляющими переходы между ними.

Некоторые расширения используют несколько имен: одно – для применения новой функциональности к коллекциям элементов DOM, а другое (или несколько других) – для дополнительных особенностей, не имеющих прямого отношения к этим коллекциям. Например, расширение Validation использует `validate` как имя основной функции, а также определяет функции `valid`, `rules` и `removeAttrs` для работы с коллекциями jQuery. Каждый раз, создавая еще одно имя, вы увеличиваете вероятность конфликтов с некоторым другим расширением, что может привести к нарушению работоспособности одного из них или обоих.

Если вам удастся ограничиться добавлением единственного имени в пространство имен jQuery и организовать взаимодействие с расширением только с его помощью, вы существенно уменьшите вероятность конфликтов с другими расширениями.

Использование единственного имени для выполнения всех возможных операций с расширением кажется труднореализуемым, из-за чего может появиться желание ввести дополнительные функции для работы с элементами. Один из способов преодолеть эти трудности и предоставить доступ к дополнительным особенностям, по крайней мере в случае коллекций элементов, состоит в том, чтобы организовать поддержку выполнения операций по их строковым именам, как описывается в разделе 4.2.6.

4.2.3. Помещайте все в объект jQuery

Включение расширения в пространство имен jQuery, напрямую или посредством точек интеграции, устраняет вероятность конфликтов с кодом из других библиотек. Во многих случаях расширение должно подключаться к определенному атрибуту jQuery, чтобы интегрироваться с другими механизмами jQuery. Кроме того, поскольку речь идет о расширении для jQuery, пользователи будут ожидать получить возможность взаимодействия с ним непосредственно через объект jQuery.

Например, расширение Debug (http://jquery.glyphix.com/), которое позволяет выводить отладочные сообщения в консоль браузера, не связано ни с какими конкретными элементами страницы и могло бы быть реализовано как отдельный пакет. Но в нем используются механизмы jQuery для упрощения реализации, поэтому оно квалифицируется как расширение для библиотеки jQuery. Его главная функция называется `log`, она подключена непосредственно к объекту jQuery и может вызываться, как показано ниже:

```
$.log('Debugging message');
```

К сожалению (если судить с позиции обсуждаемых принципов), это расширение добавляет в jQuery также функцию с именем `debug` и глобальный флаг `DEBUG`. Было бы лучше, если бы эта функция носила имя главной функции, а глобальный флаг был бы доступен как `$.log.debug`, чтобы исключить потенциальные конфликты с другими библиотеками.

Вероятность конфликтов между расширениями в пределах пространства jQuery все еще сохраняется, но использование единственной именованной точки доступа снижает риск. Кроме того, если вы столкнетесь с другим расширением, имеющим то же имя, оно наверняка будет иметь сходные цели и похожую функциональность (если не ту же самую), поэтому весьма маловероятно, что они столкнутся в одной странице.

4.2.4. Не рассчитывайте, что имя $ будет ссылаться на jQuery

jQuery – это библиотека JavaScript, такая же как, например, Prototype, MooTools и script.aculo.us. Иногда пользователю может потребоваться задействовать возможности этих библиотек в одной странице, а так как в языке JavaScript нет четкого деления кода на модули или пакеты, между библиотеками могут возникать конфликты. В частности, все упомянутые библиотеки используют идентификатор `$` как псевдоним для своих главных функций.

Разработчики jQuery знают о существовании этой проблемы и предусмотрели механизмы ее преодоления. Несмотря на возможность всегда использовать имя `jQuery` для ссылки на библиотеку, разработчики часто ленятся работать с клавиатурой и предпочитают сэкономить на нажатиях клавиш, если это возможно, особенно когда речь идет о часто используемых именах.

Чтобы создать короткую ссылку на `jQuery` и отказаться от использования имени `$`, можно вызвать функцию `noConflict`, возвращающую идентификатор `$` библиотеке, которая использовала его до загрузки jQuery, и определяющую новую переменную. В листинге 4.2 показано, как на практике выглядит создание нового синонима (`jq`) для `jQuery`.

Листинг 4.2. Устранение конфликтов между библиотеками

```
// ❶ Восстановить прежнюю переменную $
var jq = jQuery.noConflict();
// ❷ Использовать синоним
jq(document).ready(function() {
    jq('p.main')...
});
```

В главе 5 вы увидите другой механизм, гарантирующий возможность по-прежнему использовать псевдоним `$` для доступа к `jQuery` без необходимости вызывать `noConflict`.

4.2.5. Скрывайте тонкости реализации с использованием областей видимости

Как упоминалось выше, в языке JavaScript нет четкого деления кода на модули или пакеты. По умолчанию переменные объявляются в глобальном пространстве имен объекта `window`, что может привести к конфликтам с другими расширениями, которые могут пытаться изменять значения друг друга. Подобные конфликты трудно поддa-

ются выявлению, поэтому лучше сразу стараться предотвратить их возникновение.

Вам требуется применить некоторый механизм сокрытия внутренних особенностей расширения от других расширений, имеющихся в странице, чтобы гарантировать невмешательство в их работу и, что более важно, их невмешательство в работу вашего расширения. В объектно-ориентированном программировании (ООП) этот механизм называется *инкапсуляцией*. К счастью, JavaScript предоставляет похожий механизм, который называется *областью видимости*.

В языке JavaScript, определяя функцию, вы создаете новую область видимости: новый блок кода, обладающий собственным множеством имен переменных и функций. Внутри этой функции все еще можно обращаться к элементам, объявленным за пределами функции, но внешний код не имеет доступа к элементам, объявленным внутри функции. Листинг 4.3 демонстрирует эту особенность.

Листинг 4.3. Пример организации различных областей видимости переменных

```
var i = 0;          // ❶ Объявление глобальной переменной

function one() {
    i = 1;          // ❷ Ссылка на глобальную переменную
    alert(i);
}

function two(i) {  // ❸ Параметр скрывает глобальную переменную
    i = 2;
    alert(i);
}

function three() {
    var i = 3;      // ❹ Объявление локальной переменной
    alert(i);       //    скрывает глобальную переменную
}

alert(i); // 0 - глобальная переменная
one();    // 1 - глобальная переменная
two(i);   // 2 - переменная-параметр
three();  // 3 - локальная переменная
alert(i); // 1 - глобальная переменная ❺, не изменилась
```

Здесь сначала объявляется глобальная переменная `i`, и ей присваивается значение 0 ❶. Внутри функции `one` имя `i` ссылается на глобальную переменную, поэтому когда вызывается эта функция, она

изменяет и выводит значение глобальной переменной ❷. Но внутри функции two значение глобальной переменной i копируется в отдельный параметр с тем же именем, поэтому изменяется и выводится локальная копия ❸. Аналогично функция three определяет отдельную, локальную переменную i, присваивает ей значение и выводит ❹. Когда выполняется заключительный вызов alert, он выводит значение глобальной переменной i, не изменившееся с момента вызова функции one ❺. Результатом работы этого кода является вывод последовательности чисел: 0, 1, 2, 3, 1.

Чтобы защитить свой код, можно обернуть его вызовом функции, дабы создать для него новую область видимости. О том, как этот прием можно использовать при разработке своих расширений, рассказывается в разделе 5.3.2.

4.2.6. Используйте методы для доступа к дополнительной функциональности

Чтобы соблюсти принцип, требующий ограничиваться введением единственного имени в пространство имен jQuery, необходим некоторый способ предоставления доступа к дополнительной функциональности через это имя. В jQuery UI используется прием обращения к единственной функции и передаче ей имени метода, который требуется вызвать вместе с дополнительными параметрами для этого метода.

В этом случае инициализация расширения выполняется применением его главной функции непосредственно к коллекции элементов DOM, как, например, в расширении jQuery UI Tabs:

```
$('#tabs').tabs(); // С необязательными параметрами инициализации
```

А дополнительные операции с этими элементами выполняются передачей имени требуемого метода вместе с любыми параметрами:

```
$('#tabs').tabs('disable');             // Деактивировать вкладки
$('#tabs').tabs('option', {active: 2}); // Открыть третью вкладку
```

Инфраструктура поддержки расширений, описываемая в главе 5, и инфраструктура поддержки виджетов jQuery UI, описываемая в главе 8, реализуют такую возможность простым и доступным способом.

4.2.7. Возвращайте объект jQuery, если это возможно

Одной из фундаментальных особенностей jQuery является возможность составления цепочек из вызовов функций, применяемых к од-

ной и той же коллекции элементов DOM. Это позволяет записывать программный код более компактно, и пользователи ожидают, что новые функции также будут поддерживать эту возможность.

```
$('#myElement').myplugin({field: value}).show();
```

Обеспечить ее поддержку очень просто, достаточно лишь вернуть переменную `this` или ее эквивалент из главной функции расширения. Обе инфраструктуры поддержки расширений и виджетов, описываемые в главах 5 и 8, поддерживают такую возможность.

Иногда бывает желательно вернуть из экземпляра расширения какое-то определенное значение, например текущее значение обернутого элемента. Разумеется, вы вполне можете сделать это, пожертвовав возможностью добавлять в цепочку последующие вызовы других функций. Такие отклонения в обязательном порядке должны недвусмысленно отражаться в документации, чтобы пользователи знали, чего ожидать от вашего расширения.

4.2.8. Используйте функцию data для сохранения данных экземпляра

Часто бывает необходимо сохранять некоторые данные экземпляра расширения, такие как значения параметров, признак активности или ссылки на другие элементы, которыми он управляет. Такие данные должны сохраняться для каждого целевого элемента.

Для этих целей рекомендуется использовать функцию `data()` из библиотеки jQuery и добавлять все необходимые данные в виде единственного объекта, присоединенного к целевому элементу, управляемому расширением. Использование единственного имени для идентификации расширения и здесь поможет уменьшить вероятность конфликтов с другими расширениями. Данные, определяемые расширением, доступны не только из этого расширения, но и извне. Эта особенность может пригодиться для отладки с применением таких инструментов, как FireQuery (расширение FireBug для Firefox).

Например, информацию об элементе можно сохранить вызовом `data`, передав имя и значение. Значение может быть простым числом, строкой и даже объектом с собственными атрибутами:

```
$('#myElement').data('simple', 123);
$('#myElement').data('complex',
    {url: 'www.example.com', timeout: 1000, cache: true});
```

Сохраненные данные извлекаются с помощью той же функции `data`, но при этом ей нужно передать только имя:

```
var simple = $('#myElement').data('simple'); // = 123
var complex = $('#myElement').data('complex');
    // = {url: 'www.example.com', timeout: 1000, cache: true}
```

Можно также получить все данные, ассоциированные с элементом, вызвав функцию data без параметров:

```
var all = $('#myElement').data();
    // = {complex: {url: 'www.example.com', timeout: 1000, cache: true},
    //    simple: 123}
```

Кроме того, при удалении элемента из дерева DOM ассоциированные с ним данные удаляются автоматически, что исключает появление утечек памяти.

4.2.9. Предусматривайте возможность настройки

Обычно расширения реализуют некоторую определенную функциональность, но иногда может быть желательно, чтобы расширение меняло свое поведение в зависимости от некоторых условий. Пользователи всегда надеются получить возможность настройки поведения или внешнего вида расширения в соответствии со своими потребностями. Если вам удастся предугадать, что именно пожелают изменить пользователи, и предоставить соответствующий механизм настройки, это увеличит шансы вашего расширения на более широкое использование.

Например, мое расширение Datepicker предоставляет множество параметров настройки его поведения, часть которых представлена на рис. 4.3.

Очевидными кандидатами для настройки являются любые текстовые значения, отображаемые расширением. Их наверняка потребуется изменить при отображении на других языках, отличных от английского (подробнее об этом рассказывается в разделе 4.2.11). *Специальные коды* (литеральные значения, имеющие специальное значение) также являются кандидатами для включения в список настраиваемых параметров. Эти значения могут иметь определенную, предусмотренную вами область применения, но кто-то может попытаться использовать ваше расширение иным способом и пожелать изменить эти настройки.

Инфраструктура поддержки расширений, рассматриваемая в главе 5, и инфраструктура поддержки виджетов jQuery UI, рассматриваемая в главе 8, демонстрируют, как ассоциировать параметры настройки с элементом и как их использовать для изменения поведения или внешнего вида расширения.

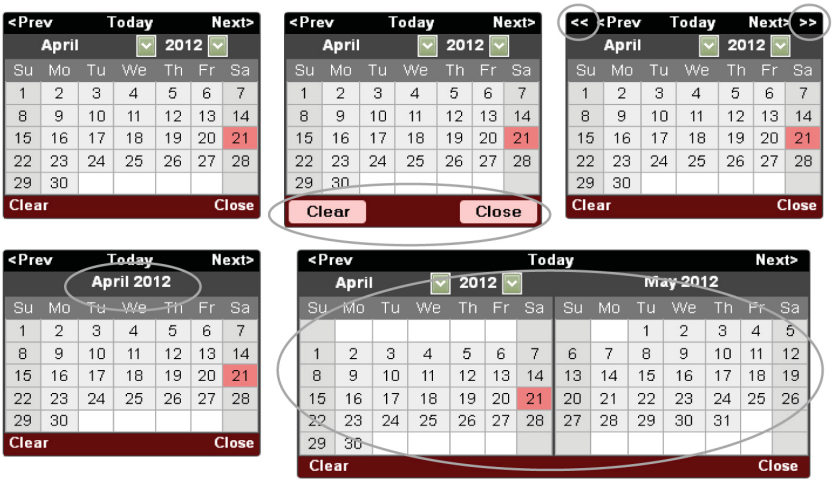


Рис. 4.3 ❖ Настройки расширения Datepicker: (верхний ряд) с настройками по умолчанию, с кнопками вместо ссылок, с элементами навигации для изменения года; (нижний ряд) отсутствует прямой выбор месяца и года, отображает сразу два месяца

Прислушивайтесь к отзывам пользователей. В них содержится ценная информация о том, как они используют расширение и что они хотели бы изменить. Попробуйте удовлетворить их пожелания через включение дополнительных параметров настройки, чтобы в следующий раз вы могли с чувством удовлетворения указать на тот или иной параметр, решающий некоторую проблему.

Не используйте параметры для оформления внешнего вида расширения. Вместо этого предусмотрите соответствующую разметку с классами, которые могут определяться во внешних файлах CSS. Такое отделение содержимого от оформления считается общепринятой практикой и существенно упрощает изменение внешнего вида расширения. Пример подобного подхода приводится в разделе 4.2.12.

4.2.10. Используйте осмысленные значения по умолчанию

Поддержка настройки поведения расширения под потребности пользователей весьма желательна, и в каждом расширении можно найти множество аспектов, пригодных для настройки. Но не переусерд-

ствуйте и не вынуждайте определять множество значений тех, кто хочет воспользоваться расширением в простеньком приложении.

Все параметры настройки должны иметь значения по умолчанию, приемлемые в большинстве случаев. Это даст пользователям возможность использовать расширение без дополнительных настроек (в действительности все настройки должны быть необязательными), действующее в стандартном режиме. В таком случае они смогут переопределить значения требуемых им параметров или изменить настройки позднее, когда в этом возникнет необходимость.

Например, мое расширение Datepicker имеет более 40 параметров (не считая параметров, отвечающих за локализацию), но ни один из них не требуется изменять, чтобы расширение могло действовать в стандартном режиме. В листинге 4.4 показаны некоторые из этих параметров.

Листинг 4.4. Параметры настройки расширения Datepicker

```
this._defaults = {
    pickerClass: '',   // Класс CSS для добавления в этот экземпляр
    showOnFocus: true, // расширения true: при получении фокуса ввода
                       // автоматически отображает календарь
    showTrigger: null, // Элемент, щелчок на котором будет приводить
                       // к выводу календаря, или none
    showAnim: 'show',  // Имя анимационного эффекта jQuery, применяемого
                       // для вывода календаря, '' - вывод без эффекта
    showOptions: {},   // Расширенные параметры для анимационного эффекта
    showSpeed: 'normal',  // Продолжительность анимационного эффекта
    popupContainer: null, // Элемент, куда будет добавляться календарь,
                          // null соответствует элементу body
    alignment: 'bottom',  // Сторона календаря для выравнивания с полем
                          // ввода: 'top' или 'bottom' - выравнивание
                          // зависит от направления письма в текущем языке,
                          // 'topLeft', 'topRight', 'bottomLeft',
                          // 'bottomRight'
    fixedWeeks: false, // true: всегда отображается 6 недель,
                       // false: отображается минимально необходимое
                       // число недель
    firstDay: 0,       // Первый день недели, 0 = воскресенье,
                       // 1 = понедельник, ...
    ...
    onDate: null,      // Функция, которая вызывается для каждой даты,
                       // отображаемой в календаре
    onShow: null,      // Функция, которая вызывается непосредственно
                       // перед выводом календаря
    onChangeMonthYear: null, // Функция, которая вызывается при выборе
                             // нового месяца/года
```

```
    onSelect: null,     // Функция, которая вызывается при выборе даты
    onClose: null,      // Функция, которая вызывается при закрытии
                        // календаря
    altField: null,     // Дополнительное поле, куда должна выводиться
                        // выбранная дата
    altFormat: null,    // Формат вывода даты для дополнительного поля,
                        // по умолчанию используется dateFormat
    constrainInput: true,    // true: разрешает вводить ограниченное
                             // подмножество символов, определяемое
                             // значением dateFormat
    commandsAsDateFormat: false, // true: применить formatDate
                                 // к навигационным ссылкам
    commands: this.commands  // Список доступных команд
                             // сюда можно добавлять новые команды
};
```

Объявление всех возможных параметров настройки и определение значений по умолчанию для них может также служить своеобразной документацией по настройке расширения.

Однако не каждое расширение может использоваться вообще без указания параметров настройки на этапе инициализации. Например, мое расширение Countdown совершенно бесполезно, если не указать ему время срабатывания, и нет никакого разумного значения по умолчанию, которое можно было бы применить в большинстве случаев.

Аналогично необходимо предусматривать оформление по умолчанию для своего расширения в виде внешнего файла CSS, опираясь на разметку и классы. Благодаря этому пользователи смогут изменить оформление, либо переопределив атрибуты элемента, либо создав собственный файл CSS.

4.2.11. Добавьте поддержку локализации

Чтобы расширить круг пользователей своего расширения, позаботьтесь о тех, кто не говорит на других языках или на других диалектах вашего языка. Любые текстовые значения, используемые в вашем расширении, должны быть сгруппированы так, чтобы их легко можно было перевести на другой язык и внедрить в расширение, дабы переопределить значения по умолчанию.

Но под локализацией содержимого понимается не только изменение текстовых значений. Не забывайте также о форматах представления дат, чисел и валют. Эти параметры также должны размещаться рядом с текстовыми значениями. Кроме того, в некоторых языках используется направление письма справа налево, и ваше расширение должно предусматривать соответствующий параметр настройки. На

рис. 4.4 изображены некоторые локализованные версии моего расширения Datepicker.

Рис. 4.4 ❖ Виджет Datepicker
на французском, японском и арабском языках

Вот некоторые различия между этими версиями, кроме очевидных изменений в текстовых значениях:

- раскрывающиеся списки выбора месяца и года располагаются в разном порядке;
- в каждом из календарей неделя начинается с разных дней: во французской версии – с понедельника, в японской – с воскресенья, в арабской – с субботы;
- в арабской версии используется письмо справа налево, в других версиях – слева направо.

Инфраструктура поддержки расширений, представленная в главе 5, включает механизм локализации расширения, упрощающий поддержку других языков. Следуя такому подходу, расширение Datepicker в настоящее время поддерживает более 70 языков, при этом переводы были предоставлены членами сообщества jQuery.

4.2.12. Реализуйте оформление внешнего вида с помощью CSS

Предоставляйте файл CSS с настройками визуального оформления вашего расширения, как показано на рис. 4.5. Прием отделения оформления от содержимого является общепринятой практикой и позволяет пользователям изменять визуальное оформление без лишних усилий.

Добавляйте классы в элементы, создаваемые и управляемые вашим расширением, чтобы обеспечить уникальность их идентификации и

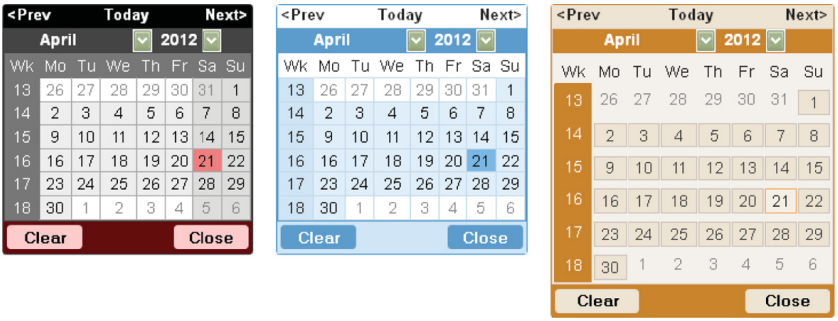


Рис. 4.5 ❖ Оформление расширения Datepicker с помощью CSS: стиль по умолчанию, стиль Redmond и стиль Humanity

оформления. Старайтесь использовать в именах классов узнаваемые префиксы, позволяющие отличать ваши классы от других, которые могут использоваться иными элементами, и уменьшающие вероятность конфликтов с существующими стилями.

Например, в листинге 4.5 приводится базовая разметка расширения Datepicker с соответствующими классами CSS. Данная разметка совместно с классами CSS, определения которых приводятся в листинге 4.6, позволяет получить оформление, как изображено на рис. 4.5 выше.

Листинг 4.5. Разметка расширения Datepicker

```
<!-- Общий контейнер для календаря -->
<div class="datepick" style="width: 218px;">
    <!-- Контейнер для элементов навигации -->
    <div class="datepick-nav">
        <!-- Навигационные ссылки -->
        <a class="datepick-cmd datepick-cmd-prev">&lt;Prev</a>
        <a class="datepick-cmd datepick-cmd-today">Today</a>
        <a class="datepick-cmd datepick-cmd-next">Next&gt;</a>
    </div>
    <!-- Строка вывода названия месяца -->
    <div class="datepick-month-row">
        <!-- Отдельно месяц -->
        <div class="datepick-month">
            ...
        </div>
    </div>
    ...
</div>
```

Листинг 4.6. Стили оформления Datepicker

```
.datepick { /* Контейнер расширения Datepicker */
    background-color: #fff;
    color: #000;
    border: 1px solid #444;
    border-radius: 0.25em;
    font-family: Arial,Helvetica,Sans-serif;
    font-size: 90%;
}
.datepick a {
    color: #fff;
    text-decoration: none;
}
.datepick-nav { /* Контейнер для элементов навигации */
    float: left;
    width: 100%;
    background-color: #000;
    color: #fff;
    font-size: 90%;
    font-weight: bold;
}
.datepick-cmd { /* Оформление навигационных ссылок */
    width: 30%;
}
.datepick-month-row { /* Оформление строки с названием месяца */
    clear: left;
}
.datepick-month { /* Отдельно название месяца */
    float: left;
    width: 15em;
    border: 1px solid #444;
    text-align: center;
}
...
```

Предусматривая параметры настройки оформления и напрямую применяя их к элементам внутри расширения, вы лишаете пользователя возможности изменять внешний вид его компонентов. Встроенные стили CSS могут переопределяться внешними, только если в определения внешних стилей будет добавлен императив `!important`, применения которого следует избегать по мере возможности.

Кроме того, в каждом элементе имеется большое число свойств, доступных для управления посредством стилей. Как вы будете выбирать, какие из них предоставлять для настройки через параметры? А что, если пользователь пожелает изменить некоторый аспект

оформления, не включенный в параметры настройки? В подобных случаях пользователям придется смешивать параметры настройки расширения с правилами CSS, что осложнит подбор оформления.

4.2.13. Тестируйте расширение в основных браузерах

И снова, чтобы получить как можно более широкое распространение, расширение должно работать во всех основных браузерах, как, например, показано на рис. 4.6. Расширение должно выглядеть и действовать одинаково на всех платформах.

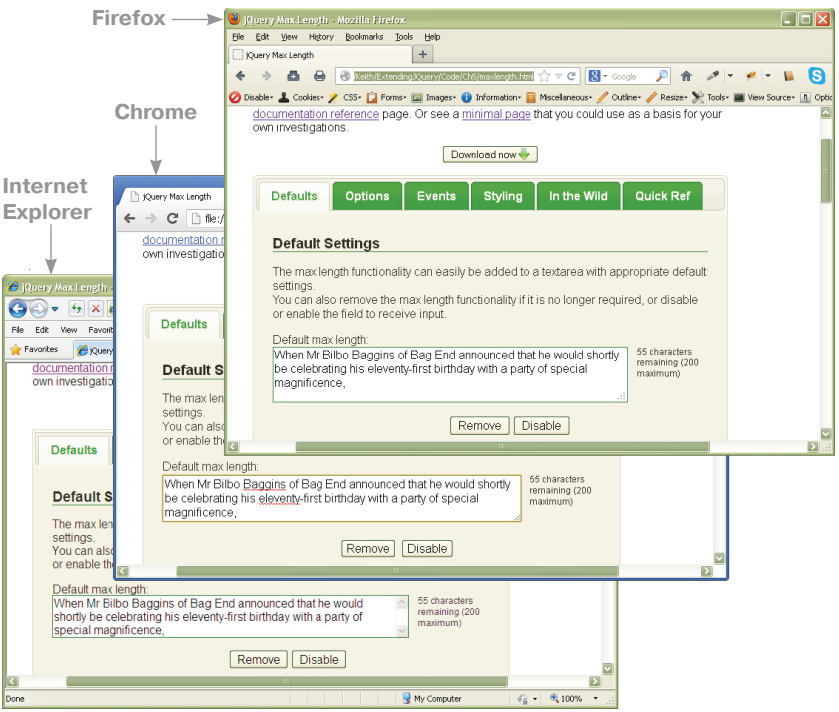


Рис. 4.6 ❖ Тестируйте свое расширение в основных браузерах, чтобы убедиться, что оно везде выглядит и действует одинаково

Для разработки я обычно использую Firefox, и получающийся код чаще всего хорошо работает также в Chrome и Safari. Internet Explorer

нередко требует особого внимания, особенно когда нужно обеспечить поддержку старых версий.

4.2.14. Создавайте комплекты повторимых тестов

Используйте инструменты автоматизированного тестирования, такие как QUnit, для создания повторимых комплектов тестов, которые легко и просто можно запускать после внесения изменений. Тесты должны охватывать все параметры, методы и функции, имеющиеся в расширении. Но инструмент QUnit не поддерживает тестирования различных визуальных аспектов расширений, поэтому вам все равно придется загружать тестовую страницу в каждом из браузеров.

На мой взгляд, создание демонстрационной страницы для расширения служит двум целям. Во-первых, она показывает потенциальным пользователям возможности расширения и, если хотите, его код. Во-вторых, она позволяет протестировать аспекты визуального представления расширения и проверить единообразие внешнего вида в разных браузерах.

Подробнее о тестировании расширений с помощью QUnit рассказывается в главе 7.

4.2.15. Создавайте демонстрационные примеры и документацию

Каким бы великолепным ни было ваше расширение, но если пользователи не поймут, как им пользоваться или как его настраивать, оно не получит широкого распространения.

Для каждого расширения должна иметься демонстрационная страница, показывающая большинство, если не все возможности расширения, как, например, страница, изображенная на рис. 4.7, демонстрирующая расширение MaxLength. Как дополнительное поощрение вы должны включить в страницу код, используемый для достижения демонстрируемого поведения, чтобы пользователи могли сразу скопировать его и вставить в свои страницы.

Также обязательно следует отразить в документации все возможности расширения. Даже при том, что они наверняка описываются в комментариях, в программном коде (по крайней мере, должны описываться), не каждый захочет погружаться в сценарий, чтобы понять, как его использовать. Создайте для расширения справочную страницу, где будут перечислены все параметры настройки, методы и другие

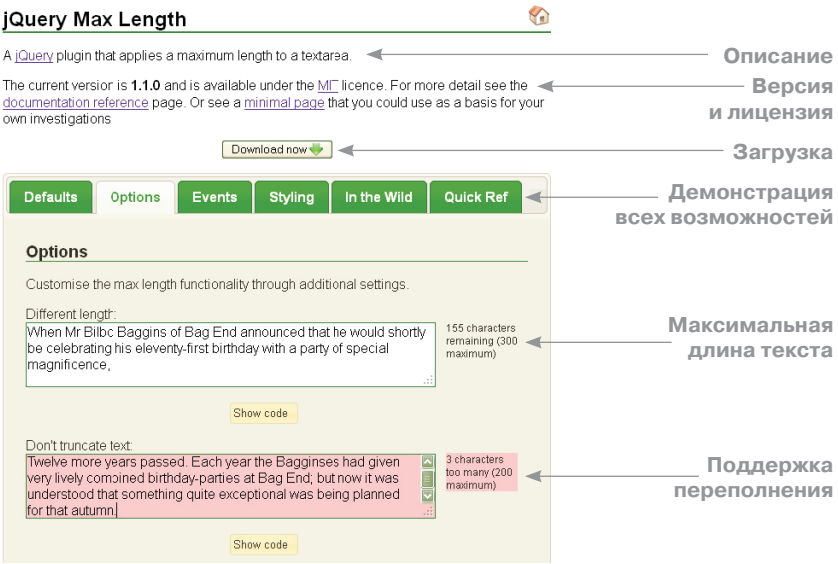


Рис. 4.7 ❖ Демонстрационная страница для расширения MaxLength, с помощью которой можно выяснить влияние разных параметров настройки

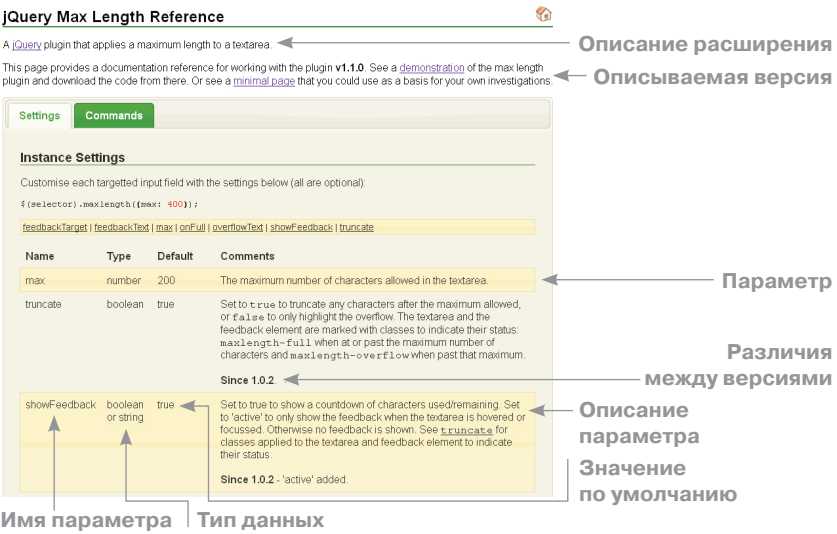


Рис. 4.8 ❖ Документация с описанием параметров настройки расширения MaxLength

функции. На рис. 4.8 показано, как выглядит фрагмент документации с описанием расширения MaxLength.

Для каждого параметра должны приводиться его имя, тип или типы данных, значение по умолчанию (чтобы пользователь знал, чего можно ожидать) и описание назначения. Если параметр может принимать ограниченное множество значений, перечислите и объясните каждое из них. Если параметр имеет сложную структуру, такую как объект, подробно опишите ее строение, включая внутренние атрибуты, их типы и назначение. Обязательно предоставляйте примеры кода, когда параметр сложнее, чем строка, число или логическое значение.

Аналогично для каждого метода должны приводиться его имя и входные параметры вместе с их типами и назначением. Отметьте необязательные входные параметры. Укажите, какое значение возвращается методом, и особо подчеркните, если метод разрывает цепочку вызовов.

Это нужно знать

Тщательно планируйте архитектуру расширения и приемы взаимодействия с ним до начала разработки.
Всегда, когда это возможно, расширяйте базовые возможности библиотеки с помощью расширений.
Предотвращайте конфликты имен и защищайте свой код с применением областей видимости.
Предусматривайте параметры настройки в своих расширениях, чтобы сделать их более гибкими, и назначайте им осмысленные значения по умолчанию.
Тестируйте свои расширения, чтобы убедиться в их работоспособности.
Пишите документацию и создавайте демонстрационные страницы, чтобы помочь другим начать использовать ваши расширения.

4.3. В заключение

Расширения для jQuery могут практически все, что угодно, от простого изменения классов элементов и обработки событий до реализации селекторов, анимационных эффектов, полнофункциональных графических виджетов и средств удаленного доступа. Выбор области приложения ваших усилий может оказаться нелегким.

Планируйте внешний вид и поведение расширения. Представьте, как оно будет взаимодействовать с библиотекой jQuery, пользователем и другими элементами страницы. Подумайте, как оно могло бы настраиваться пользователем. Но не теряйте из виду главную цель, не стремитесь включить в него все, что только сможете придумать.

Чтобы обеспечить корректное взаимодействие расширения с jQuery и любыми другими библиотеками JavaScript, необходимо следовать определенным правилам. Принципы проектирования, описанные здесь, содержат ряд рекомендаций, которые в равной степени подходят для создания любых расширений независимо от их типа.

В следующей главе и в главе 8 вы увидите, как реализованы эти принципы в двух инфраструктурах поддержки расширений для основной библиотеки jQuery и виджетов для jQuery UI.

Глава 5

Расширения коллекций

Эта глава охватывает следующие темы:

- определение расширений коллекций;
- использование инфраструктуры поддержки расширений;
- применение принципов проектирования;
- создание законченного расширения коллекций.

Теперь, когда вы познакомились с теорией проектирования и реализации расширений, можно перейти к практике. Выражаясь более конкретно, в этой главе мы создадим относительно простое расширение, реализующее полезные возможности, и вместе с тем достаточно сложное, чтобы на его примере можно было показать применение приемов, знание которых пригодится вам при создании любых расширений.

Расширение, которое мы создадим, дополняет существующую функциональность, предоставляемую браузером. Обычные поля ввода имеют атрибут `maxlength`, позволяющий ограничить количество символов, которое можно ввести в поле. Это помогает обеспечить соответствие ограничениям, накладываемым базами данных и другими механизмами хранения данных. Но многострочное поле ввода `textarea` не имеет такого атрибута и никак не ограничивает ввод. Чтобы исправить этот недостаток, мы создадим расширение для управления ограничением объема вводимого текста и реализуем поддержку обратной связи с пользователем.

5.1. Что такое расширения коллекций?

Как уже говорилось в главе 3, библиотека jQuery обычно работает по принципу «выбирай и действуй» – вы находите элементы, которые вас интересуют, либо непосредственно, с помощью селекторов и фильтров, либо посредством обхода дерева DOM, и затем выполняете

требуемые операции. Например, чтобы скрыть все абзацы с классом note, можно использовать следующий код:

```
$('p.note').hide();
```

Такие операции я называю *расширениями коллекций* – они оперируют коллекциями элементов DOM, завернутых в объект jQuery. Сторонние расширения для библиотеки jQuery в большинстве своем относятся именно к этому типу.

Подключение расширения коллекций выполняется через точку интеграции $.fn, путем добавления атрибута с именем нового расширения и присваивания ему функции, реализующей функциональность расширения. В действительности имя $.fn – это псевдоним для $.prototype, стандартного механизма в языке JavaScript, используемого для добавления свойств и методов ко всем экземплярам объекта данного типа. Добавление новой функции в $.fn автоматически делает ее доступной для всех объектов jQuery, таких как коллекции элементов, создаваемые в процессе выбора или последовательного обхода дерева DOM. Такие функции вызываются в контексте текущего экземпляра jQuery и, соответственно, имеют доступ к коллекции элементов, которой управляет данный экземпляр.

5.2. Инфраструктура поддержки расширений

Даже при том, что каждое расширение можно разрабатывать что называется «с нуля», есть смысл повторно использовать некоторый код, потому что все расширения взаимодействуют с библиотекой jQuery практически одинаково. С этой целью я создал инфраструктуру поддержки расширений, которую с успехом использую для своих расширений. Она реализует механизмы, общие для всех расширений коллекций, упрощает добавление новых параметров настройки и методов, а также обеспечивает следование руководящим принципам, описанным в предыдущей главе.

5.2.1. Расширение MaxLength

Расширение, которое мы создадим в этой главе, ограничивает максимальную длину текста в поле textarea и действует подобно атрибуту maxlength полей ввода однострочного текста. С настройками по умолчанию вызов расширения выглядит, как показано ниже:

```
$('#text1').maxlength();
```

Для настройки расширения в вызов его функции следует передать дополнительный параметр:

```
$('#text1').maxlength({max: 400});
```

Помимо ограничения длины текста, который можно ввести, расширение предоставляет обратную связь, сообщая максимальную длину текста и количество символов, которые еще можно ввести (рис. 5.1).

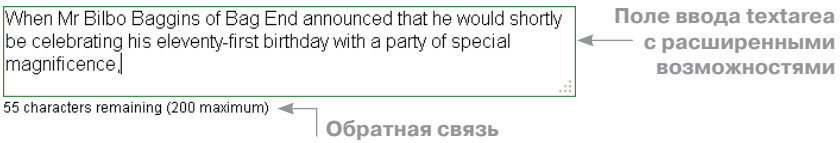


Рис. 5.1 ❖ Расширение MaxLength в действии

Кроме того, расширение предупреждает о достижении максимальной длины введенного текста, не препятствуя вводу дополнительных символов. Благодаря этому пользователь будет знать, что он должен сократить введенный текст, и при этом расширение не вынуждает его прерывать ход мыслей на полпути. Вместо этого пользователю предоставляется возможность вернуться назад и отредактировать текст самостоятельно. Предупреждение появляется, только когда длина текста превысит установленный максимум.

Расширение поддерживает возможность полностью подавить вывод предупреждения, однако дополнительная информация будет вызывать у пользователя более благоприятные ощущения. Кроме того, имеется возможность выводить предупреждение, только когда поле ввода активно, например при наведении указателя мыши или когда поле ввода получает фокус ввода. Дополнительно предусмотрена возможность информирования пользователя через возбуждение события.

Расширение MaxLength поддерживает принцип *прогрессивного наращивания возможностей*. Оно улучшает восприятие пользователя при вводе текста в поле с ограничением. В браузерах с отключенной поддержкой JavaScript возможность ввода текста сохраняется, а ограничение длины будет применено на сервере (подобные ограничения на сервере должны действовать в любом случае).

5.2.2. Устройство расширения MaxLength

Расширение состоит из объекта, который содержит несколько функций, реализующих дополнительные возможности, и позволяет пользователю инициализировать и управлять этими возможностями.

В листинге 5.1 представлены общая структура расширения и примеры вызова его функций в течение жизненного цикла.

Листинг 5.1. Структура расширения

```
function MaxLength() {
    // Параметры настройки по умолчанию
    this._defaults = {...};
}

$.extend(MaxLength.prototype, {
    // Переопределяет настройки по умолчанию
    setDefaults: function(options) {...},
    // Инициализация
    _attachPlugin: function(target, options) {...},
    // Возвращает/устанавливает параметры настройки
    _optionPlugin: function(target, options, value) {...},
    // Возвращает текущую длину
    _curLengthPlugin: function(target) {...},
    // Накладывает ограничение на длину
    _checkLength: function(target) {...},
    // Активизирует элемент
    _enablePlugin: function(target) {...},
    // Деактивизирует элемент
    _disablePlugin: function(target) {...},
    // Отключает дополнительную функциональность от элемента
    _destroyPlugin: function(target) {...}
});

// Интеграция с библиотекой jQuery
$.fn.maxlength = function(options) {...};

// Объект-одиночка (singleton)
var plugin = $.maxlength = new MaxLength();
```

Объект-одиночка расширения (`plugin`) создается сразу после загрузки кода расширения. Пользователь взаимодействует с этим объектом непосредственно через ссылку в объекте jQuery (`$.maxlength`) или через функцию интеграции (`$.fn.maxlength`), позволяющую применять функциональность расширения к коллекции элементов, полученной с помощью селекторов.

Параметры настройки по умолчанию для всего расширения находятся в атрибуте `_defaults` объекта-одиночки. Переопределить эти параметры можно передачей новых значений в вызов функции `setDefaults`. Они автоматически будут наследоваться всеми вновь создаваемыми экземплярами расширения.

Подключить расширение к одному или нескольким элементам можно применением функции интеграции к выбранному множеству

элементов. Она передает управление функции _attachPlugin, которая инициализирует целевые элементы и затем вызывает _optionPlugin, передавая ей текущие значения параметров настройки (либо принятые по умолчанию, либо указанные в вызове функции инициализации) для применения к данному экземпляру:

```
$('#text1').maxlength({max: 400});
```

Функция _optionPlugin вызывается также в ответ на обращение к методу option с целью изменить значения параметров. Текущие значения параметров и другие внутренние данные привязываются к соответствующему элементу (или элементам).

```
$('#text1').maxlength('option', 'onFull', alertMe);
```

В конечном итоге все пути приводят к функции _checkLength (в ходе инициализации, при изменении параметров, когда пользователь отпускает нажатую клавишу или когда поле получает фокус ввода), которая реализует ограничение длины и может предупредить пользователя, если поле окажется заполнено или переполнено.

Чтобы запретить доступ к полю ввода, можно обратиться к методу disable, в результате чего будет вызвана функция _disablePlugin. Аналогично восстановить доступ к полю можно обращением к методу enable, который вызовет функцию _enablePlugin:

```
$('#text1').maxlength('disable');
```

В любой момент вызовом метода curLength можно получить число введенных символов и оставшихся до максимального значения. Вычисленные значения возвращаются соответствующей функцией _curLengthPlugin:

```
var lengths = $('#text1').maxlength('curLength');
```

Если надобность в расширении отпала, его можно удалить обращением к методу destroy, который, в свою очередь, вызовет функцию _destroyPlugin. Эта функция отменит все действия, выполненные функциями инициализации и настройки параметров, и вернет соответствующие элементы в исходное состояние:

```
$('#text1').maxlength('destroy');
```

В следующих разделах более подробно описывается, как работают эти функции и как они взаимодействуют между собой, обеспечивая возможности MaxLength.

5.3. Определение собственного расширения

Прежде чем приступить к реализации конкретных функциональных особенностей расширения, необходимо выполнить несколько шагов:

- выбрать имя для расширения;
- защитить свой код от внешнего мира JavaScript, и наоборот;
- определить объект-одиночку, обеспечивающий доступ к настройкам и функциям.

5.3.1. Выбор имени

Каждому расширению необходимо присвоить имя для его идентификации и отделения от других расширений. Имя должно отражать назначение расширения и выбираться так, чтобы его можно было использовать повсюду, в соответствии с принципом, требующим *ограничиваться единственным именем и использовать только его для взаимодействий с расширением*. В программном коде расширения и в документации с его описанием допускается использовать немного разные имена, но они должны быть близки друг к другу.

Имейте в виду, что имя является точкой доступа к расширению. Если выбрать имя, совпадающее с именем другого расширения, эти два расширения нельзя будет использовать совместно в одной странице. С другой стороны, расширения, обладающие одинаковыми именами, скорее всего, будут предоставлять похожие, если не одинаковые функциональные возможности, поэтому весьма маловероятно, что кому-то потребуется использовать два таких расширения одновременно.

Для данного расширения я выбрал имя `maxlength`, потому что оно точно отражает назначение расширения. Оно не слишком длинное и не слишком короткое. В документации на расширение можно ссылаться по имени MaxLength.

В соответствии с соглашениями, общепринятыми в сообществе jQuery, имя должно состоять только из символов нижнего регистра. Кроме того, код расширения должен храниться в файле с именем `jquery.<имя_расширения>.js`. Имена сопутствующих файлов должны следовать этому же шаблону. Для данного расширения программный код будет храниться в файле `jquery.maxlength.js`, а определения стилей CSS – в файле `jquery.maxlength.css`.

5.3.2. Инкапсуляция

Два руководящих принципа – *сокрытия тонкостей реализации с использованием областей видимости* и *отказа от предположения, что имя $ будет ссылаться на jQuery,* – могут быть реализованы с применением типового кода, представленного в листинге 5.2. Этот код, защищающий расширение от внешнего мира JavaScript, реализует парадигму, известную в объектно-ориентированном программировании как *инкапсуляция*.

Листинг 5.2. Инкапсуляция расширения

```
// ❶ Объявление анонимной функции
(function($) { // Скрывает из видимости внешнего кода,
               // устраняет конфликты с идентификатором $
    ... здесь располагается остальной код
})(jQuery); // ❷ Немедленный вызов функции
```

Сначала создается анонимная функция, играющая роль новой области видимости ❶, – переменные и функции, объявленные внутри нее, не будут доступны извне. Благодаря этому можно следовать любым соглашениям об именовании, не волнуясь о конфликтах с внешним кодом или с другими расширениями. Все, что должно быть доступно внешнему коду, экспортируется посредством самого объекта `jQuery`.

Объявление функции-обертки заключается в круглые скобки, чтобы сделать его доступным для использования, и затем она немедленно вызывается ❷. В качестве параметра функции передается объект `jQuery`. Обратившись к заголовку объявления функции, можно заметить, что она принимает единственный параметр с именем `$`. Благодаря этой хитрости объект `jQuery` отображается в параметр с именем `$`, под которым его можно использовать внутри функции для ссылки на объект `jQuery`, не опасаясь, что оно окажется подменено какой-нибудь другой библиотекой JavaScript.

> Возможно, вы обращали внимание, что большая часть кода jQuery обернута обратным вызовом `$(document).ready(function() {...})` или более короткой его формой `$(function() {...})`. Это гарантирует, что код не будет выполнен, пока дерево DOM не станет доступно для использования. Вам не нужно обертывать код своего расширения подобным образом – функция расширения должна быть выполнена немедленно сразу после загрузки, чтобы подготовить расширение к началу инициализации библиотеки jQuery. Но если расширению потребуется выполнить какие-то операции с деревом DOM, его инициализацию следует отложить до момента, пока он не потребуется, – обычно до момента применения расширения к множеству элементов – или обернуть собственной функцией обратного вызова `document.ready`.

5.3.3. Использование объекта-одиночки

Чтобы упростить взаимодействия с расширением и организовать нечто вроде центрального репозитория информации и функциональных возможностей, я использую шаблон проектирования «Одиночка» (singleton), согласно которому единственный экземпляр объекта является глобальной точкой доступа. В дополнение к определениям возможностей объекта, в виде внутренних функций, он также содержит константы и значения, используемые всеми применениями расширения к элементам страницы. Определение такого объекта-одиночки приводится в листинге 5.3.

Листинг 5.3. Определение объекта-одиночки, управляющего расширением

```
/* Диспетчер расширения MaxLength. */
// ❶ Объявление класса на JavaScript
function MaxLength() {
    // ❷ Создать массив с региональными настройками
    this.regional = [];   // Доступные региональные настройки,
                          // индексируется кодом языка
    this.regional[''] = { // Региональные настройки по умолчанию
        feedbackText: '{r} characters remaining ({m} maximum)',
        ... Прочие региональные настройки
    };
    // ❸ Объявить параметры со значениями по умолчанию
    this._defaults = {
        max: 200,       // Максимальная длина
        ... Прочие настройки по умолчанию
    };
    // ❹ Объединить регион по умолчанию со значениями по умолчанию
    $.extend(this._defaults, this.regional['']);
}

// ❺ Определить другие константы и функции
$.extend(MaxLength.prototype, {
    ...
});

/* Инициализировать функциональность maxlength. */
// ❻ Создать экземпляр объекта-одиночки
var plugin = $.maxlength = new MaxLength(); // Единственный экземпляр
```

Конструкции, напоминающие классы, в языке JavaScript определяются как функции ❶. Имя этой функции невидимо извне из-за использования новой области видимости, реализованной в предыдущем разделе, поэтому оно не обязательно должно совпадать с именем

расширения. В действительности вы можете выбрать имя, более очевидное для вашего кода.

Функция может иметь собственное множество внутренних полей и вложенных функций, определяющих данные и поведение расширения. Переменная `this` ссылается на текущий экземпляр «класса». Наиболее важной частью здесь является объявление параметров по умолчанию, управляющих поведением расширения ❸. В идеале эти параметры должны определять все настройки, необходимые для применения расширения к элементу и его функционирования в стандартном режиме. Ваши пользователи смогут переопределить любые из этих параметров при инициализации расширения для управления их собственными элементами.

Чтобы упростить локализацию параметров настройки, определяется ❷ и инициализируется массив локализованных параметров для языка по умолчанию (английский). Эти настройки затем объединяются с другими настройками по умолчанию, как только они будут определены ❹. Если будут определены локализованные настройки для другого языка, их можно применить, как показано ниже:

```
$('#text1').maxlength($.maxlength.regional['fr']);
```

Подробнее о локализации расширения будет рассказываться в разделе 5.5.2.

Затем определяются дополнительные константы и внутренние функции, и добавляются в прототип (атрибут `prototype`) функции ❺. Эти функции реализуют возможности расширения. Некоторые из них являются общими для всех расширений, а остальные определяют конкретную функциональность.

Наконец, чтобы сделать объект-одиночку доступным для внешнего кода, создается единственный его экземпляр и присваивается атрибуту объекта `jQuery` (на который ссылается псевдоним `$`) ❻ в соответствии с принципом *размещения всего необходимого в объекте jQuery*. Обратите внимание, что при этом используется единственное имя, выбранное для расширения. Дополнительно здесь же создается локальная переменная `plugin`, используемая для ссылки на расширение внутри модуля и упрощающая повторное использование инфраструктуры поддержки в реализациях других расширений.

Этот прием облегчает доступ к константам, переменным и функциям расширения даже в функциях обратного вызова, которые могут получать разные контексты. О том, как пользоваться этим приемом, рассказывается ниже в этом разделе.

5.4. Применение к элементам

Чтобы применить функциональность расширения к элементу страницы, необходимо определить функцию, которая позволит библиотеке jQuery вызывать ваш код. Все расширения, воздействующие на коллекции элементов, должны подключаться к точке интеграции `$.fn`. Однако ситуация усложняется, когда необходимо предусмотреть обработку методов и функций чтения.

В связи с этим вам необходимо узнать:

- как применить расширение к одному или более элементам;
- как инициализировать расширение в процессе применения к элементу;
- как обрабатывать имена методов, передаваемых расширению для выполнения дополнительных операций;
- как возвращать запрошенные значения из расширения.

Что ж, начнем с самых основ.

5.4.1. Простое подключение

В библиотеке jQuery имеется точка интеграции для подключения расширений коллекций, упрощающая их применение к группам элементов, полученных в процессе выбора и/или обхода дерева DOM. В листинге 5.4 показано, как использовать эту точку интеграции для включения расширения в работу.

Листинг 5.4. Применение расширения к элементу

```
/* Применить функциональность maxlength к множеству выбранных элементов.
   @param  options  (object) новые настройки для использования
                             в экземплярах (необязательный)
   @return (jQuery) для поддержки составления цепочек вызовов */
// ❶ Объявление главной функции расширения
$.fn.maxlength = function(options) {
    // ❷ Для поддержки цепочек вызовов в jQuery
    return this.each(function() {
        // ❸ Инициализировать каждый элемент
        plugin._attachPlugin(this, options || {});
   });
};
```

Функция с именем расширения подключается к точке интеграции `$.fn` ❶. Эта функция принимает один аргумент – объект со значениями параметров настройки, переопределяющими значения по умолчанию, которые влияют на поведение расширения. Если аргумент опущен, все параметры получают значения по умолчанию. Так как

объект jQuery (под псевдонимом $) является глобальной переменной, вы всегда сможете получить доступ к своему расширению через это определение. Поскольку подключение производится к точке интеграции $.fn, библиотека jQuery будет знать, что ваше расширение должно применяться к коллекциям элементов, и передавать эту коллекцию вашей функции в виде переменной this.

> Переменная this ссылается на коллекцию элементов, полученную в результате выбора с применением селекторов и/или обхода дерева DOM. Она уже является объектом jQuery, и ее не требуется обертывать другим вызовом jQuery.

Чтобы сохранить действие одной из ключевых особенностей jQuery и обеспечить следование принципу, требующему *возвращать объект jQuery, если это возможно*, необходимо всегда возвращать переменную this из своих функций ❷. Это позволяет составлять цепочки вызовов для обработки коллекций, и такое поведение ожидается пользователями расширений. Например:

```
$('#text1').maxlength().change(function() {...});
```

Так как цель расширения состоит в том, чтобы обработать каждый элемент в коллекции, с этой целью часто используется функция each, и все элементы обрабатываются по отдельности. Возвращаемым значением функции each является оригинальная коллекция, поэтому его можно возвращать из функции расширения непосредственно и тем самым поддержать возможность составления цепочек вызовов.

Наконец, для применения функциональности расширения вызывается функция объекта-одиночки (plugin) – функция _attachPlugin ❸. В качестве параметров ей передаются текущий элемент (this) и набор параметров настройки options. Так как аргумент options является необязательным, он может быть не определен. Чтобы упростить последующий код, необходимо гарантировать передачу пустого объекта с помощью конструкции options || {}, если текущий аргумент options не определен. В действительности этот код оценивает первое выражение (options) и возвращает его, если оно имеет «истинное» значение (не является значением undefined или false, пустой строкой или нулем). В противном случае вычисляется второе выражение и возвращается новый пустой объект.

5.4.2. Инициализация расширения

Функция _attachPlugin инициализирует расширение для применения к конкретному элементу страницы, как показано в листинге 5.5. Целевой элемент передается функции в первом параметре, а любые

настройки – во втором. Сюда можно добавить дополнительные операции, выполняющиеся однократно, которые управляют состоянием расширения применительно к данному элементу и не зависят от значений параметров настройки.

Листинг 5.5. Инициализация расширения перед применением к элементу

```
/* Применяет функциональность расширения maxlength к элементу textarea.
   @param  target   (element) целевой элемент
   @param  options  (object)  настройки для данного экземпляра */
_attachPlugin: function(target, options) {
    target = $(target);
    // ❶ Не инициализировать расширение повторно
    if (target.hasClass(this.markerClassName)) {
        return;
    }
    // ❷ Создать объект с настройками
    var inst = {options: $.extend({}, this._defaults),
        feedbackTarget: $([])};
    // ❸ Добавить класс-метку
    target.addClass(this.markerClassName).
        // ❹ Сохранить настройки в элементе
        data(this.propertyName, inst).
        // ❺ Добавить обработчики событий
        bind('keypress.maxlength', function(event) {
            if (!inst.options.truncate) {
                return true;
            }
            var ch = String.fromCharCode(
                event.charCode == undefined ?
                    event.keyCode : event.charCode);
            return (event.ctrlKey || event.metaKey || ch == '\u0000' ||
                $(this).val().length < inst.options.max);
        }).
        bind('keyup.maxlength', function() {
            plugin._checkLength($(this));
        });
    // ❻ Добавить настройки в элемент
    this._optionPlugin(target, options);
},
```

В первую очередь функция проверяет, не был ли данный элемент инициализирован прежде ❶, и не выполняет последующие операции, если элемент уже инициализирован. Элементы должны инициализироваться только один раз, так как было бы нежелательно подключать к ним множество обработчиков событий или многократно выполнять по сути однократные операции. Достигается это за счет проверки при-

сутствия определенного класса в элементе. Присваивание класса выполняется одним из первых действий в процессе инициализации ❸.

Чтобы дать возможность отслеживать состояние расширения, создается объект с настройками ❷, характерными для данного элемента, включая любые настройки, определяемые пользователем. Для этого сначала создается пустой объект `{}`, который затем дополняется настройками по умолчанию. Если бы здесь использовался объект `_defaults` непосредственно, любые изменения в настройках применялись бы к этому объекту и оказывали влияние на последующие применения расширения. Пользовательские настройки применяются в функции `_optionPlugin` ❻, которая обрабатывает изменения в параметрах.

Объект с данными сохраняется в элементе с использованием функции `data` ❹ (в соответствии с принципом *использования функции data для сохранения данных экземпляра*). Постоянное имя, определяемое внутри этого модуля, – опять же со значением `maxlength` – упрощает доступ к данным из любого места в расширении. То есть состояние расширения в каждом конкретном элементе можно извлекать в любых других функциях, имеющихся в расширении. При удалении элемента из дерева DOM объект с данными будет удален автоматически.

В процессе применения расширения к элементам выполняются также другие операции, не зависящие от значений параметров настройки. В данном случае производится подключение обработчиков событий `keypress` и `keyup` к целевому элементу `textarea` ❺, чтобы с их помощью проверять число введенных символов при любых изменениях. При создании обработчиков событий внутри расширений к именам событий всегда следует добавлять пространство имен (в данном случае `maxlength`). Это позволяет легко удалять обработчики, добавляемые расширениями, не затрагивая обработчиков, установленных извне.

В заключение вызывается функция `_optionPlugin`, чтобы применить любые параметры настройки к расширению ❻. Подробнее эта функция рассматривается в разделе 5.5.3.

5.4.3. Вызов методов

В разделе 5.4.1 было показано, как инициализируются расширения коллекций элементов DOM. Но в соответствии с принципами *объявления только одного имени* и *использования методов для доступа к дополнительной функциональности* необходимо предусмотреть некоторый способ предоставления доступа к дополнительным возможностям. Например, функции расширения можно передавать

строковое значение, определяющее операцию, которую она должна выполнить. Типичными примерами могут служить операции активации (enabling) и деактивации (disabling) элементов пользовательского интерфейса, изменения или извлечения параметров настройки. В последнем случае возникает необходимость передачи дополнительных значений, помимо названия операции, чтобы указать имя параметра и его новое значение.

Большая часть кода, реализующего эти возможности (листинг 5.6), является достаточно типичной и может использоваться во всех расширениях коллекций.

Листинг 5.6. Вызов методов расширения

```
/* Применяет функциональность расширения maxlength к набору элементов.
   @param  options  (object) новые параметры настройки для выбранных
                             экземпляров (необязательный) или
                    (string) имя метода (необязательный)
   @return (jQuery) для поддержки составления цепочек вызовов */
$.fn.maxlength = function(options) {
    // ❶ Извлечь аргументы со второго до последнего
    var otherArgs = Array.prototype.slice.call(arguments, 1);
    return this.each(function() {
        // ❷ Вызов метода?
        if (typeof options == 'string') {
            // ❸ Проверить наличие метода с таким именем
            if (!plugin['_' + options + 'Plugin']) {
                // ❹ Возбудить исключение, если метод не найден
                throw 'Unknown method: ' + options;
            }
            // ❺ Вызвать метод
            plugin['_' + options + 'Plugin'].
                apply(plugin, [this].concat(otherArgs));
        }
        else {
            plugin._attachPlugin(this, options || {});
        }
    });
};
```

Так как количество параметров функции расширения неизвестно заранее и зависит от вызываемого метода, для доступа к любым аргументам, следующим за первым, используется стандартная в языке JavaScript переменная `arguments`. Функция `slice` класса `Array` извлекает элементы из массива `arguments` и копирует их в другую переменную ❶. В результате в `otherArgs` оказывается массив, содержащий все параметры, переданные функции `maxlength`, кроме первого, который, как предполагается, является именем метода.

Функция выполняет итерации по всем элементам DOM в коллекции и обрабатывает их по отдельности. Внутри цикла проверяется тип аргумента `options`, чтобы определить, представляет ли он имя метода ❷. В обычном вызове, выполняющем инициализацию, аргумент `options` может отсутствовать или быть объектом со значениями параметров настройки. Если он является строкой, его следует обработать как имя метода.

Чтобы избежать непредсказуемых ошибок, проверяется возможность вызова указанного метода. В соответствии с соглашениями, чтобы отобразить имя метода в имя фактической функции, к нему нужно добавить префикс в виде символа подчеркивания (_) и стандартное окончание `Plugin` ❸. Если функция с таким именем отсутствует, возбуждается исключение ❹. Например, метод `option` будет отображен в функцию `_optionPlugin`.

> Я использую символ подчеркивания (_) в качестве префикса, чтобы показать, что функция предназначена исключительно для использования внутри расширения, но сам язык JavaScript не предусматривает никаких ограничений для таких имен. Функции, предназначенные для непосредственного использования, не имеют символа подчеркивания в их именах. Как дополнительный признак внутренних функций, играющих роль реализаций методов, чтобы отличать их от обычных внутренних функций, я добавляю стандартное окончание к их именам. Таким образом, функция `setDefaults` предназначена для непосредственного использования, функция `_curLengthPlugin` является реализацией метода `curLength`, а функция `_checkLength` просто предназначена для внутреннего использования.

Если функция, соответствующая методу, существует, она вызывается с помощью стандартной функции `apply`, которая гарантирует передачу объекта-одиночки (первый параметр) в качестве контекста для вызываемой ею функции ❺. Любые дополнительные параметры, переданные функции расширения, а теперь хранящиеся в переменной `otherArgs`, объединяются с текущим элементом в массив и превращаются в полный комплект параметров вызываемого метода.

Реализации самих методов будут представлены в последующих разделах. Теперь, когда в расширение добавлена возможность вызова методов, необходим некоторый механизм передачи возвращаемых ими значений, о чем рассказывается в следующем разделе.

5.4.4. Методы чтения

Большинство методов вызываются с целью выполнить некоторые операции над указанными элементами, но есть и такие, которые должны возвращать некоторые значения, например: текущее коли-

чество символов и количество символов, оставшееся до достижения предела, установленного расширением. Такие методы отрицательно влияют на возможность составления цепочек вызовов с целью применения множества операций к одной и той же коллекции элементов, широко используемой в библиотеке jQuery, потому что они возвращают значения, не являющиеся текущей коллекцией элементов.

В листинге 5.7 представлен код вызова метода чтения, большую часть которого можно использовать во всех расширениях коллекций.

Листинг 5.7. Применение расширения к элементу

```
// Список методов, возвращающих значения и не допускающих
// добавления других методов в цепочку после них
var getters = ['curLength']; // ❶ Список методов чтения

/* Определяет, является ли метод методом чтения и не допускающим
   добавления других методов в цепочку после него.
   @param   method     (string, необязательный) имя метода
   @param   otherArgs  ([], необязательный) любые аргументы метода
   @return  true, если метод является методом чтения,
            false, если нет */
// ❷ Проверка на принадлежность к методам чтения
function isNotChained(method, otherArgs) {
    if (method == 'option' && (otherArgs.length ==0 ||
            (otherArgs.length == 1 && typeof otherArgs[0] == 'string'))) {
        return true;
    }
    return $.inArray(method, getters) > -1;
}

/* Применяет функциональность расширения maxlength к набору элементов.
@param  options  (object) новые параметры настройки для выбранных
                          экземпляров (необязательный) или
                 (string) имя метода (необязательный)
@return (jQuery) для поддержки составления цепочек вызовов или
        (любой) как возвращаемое значение метода */
$.fn.maxlength = function(options) {
    var otherArgs = Array.prototype.slice.call(arguments, 1);
    // ❸ Если это метод чтения...
    if (isNotChained(options, otherArgs)) {
        // ❹ ...вернуть значение метода
        return plugin['_' + options + 'Plugin'].
            apply(plugin, [this[0]].concat(otherArgs));
    }
    return this.each(function() {
        if (typeof options == 'string') {
            if (!plugin['_' + options + 'Plugin']) {
                throw 'Unknown method: ' + options;
            }
```

```
            plugin['_' + options + 'Plugin'].
                apply(plugin, [this].concat(otherArgs));
        }
        else {
            plugin._attachPlugin(this, options || {});
        }
    });
};
```

Вам потребуется явно определить список методов чтения ❶, иначе они не смогут быть идентифицированы. Методы чтения обрабатываются отдельно от других методов, чтобы вернуть их значения вызывающей программе.

Так как метод `option` имеет двойственную природу – он может извлекать или изменять значение параметра настройки, его необходимо обрабатывать отдельно. Функция `isNotChained` ❷ интерпретирует этот метод особо (опираясь на количество параметров), а для других методов проверяет наличие имени метода в списке, объявленном выше, и возвращает `true`, если метод определяется как метод чтения.

Метод, идентифицированный как метод чтения ❸, вызывается немедленно, а его значение возвращается как результат работы функции расширения ❹. Подобно другим методам, методы чтения вызываются с помощью функции `apply`. Им передаются: объект-одиночка в качестве контекста, первый элемент в коллекции и любые дополнительные параметры (`otherArgs`). Методам чтения передается только первый элемент коллекции, потому что вернуть можно только одно значение (нельзя несколько раз вернуть результаты нескольких вызовов метода чтения для нескольких элементов); этот шаблон программирования соответствует стандартной практике, принятой в библиотеке jQuery и используемой в методах чтения, таких как `attr`, `css` и `val`.

Вызываются такие методы примерно так:

```
var counts = $('#text1').maxlength('curLength');
```

Реализация функции `_curLengthPlugin` будет представлена в разделе 5.7.1.

5.5. Параметры настройки

Наличие параметров настройки позволяет управлять поведением расширения. Среди руководящих принципов есть два, имеющих прямое отношение к настройкам: *«предусматривайте возможность*

настройки» и *«используйте осмысленные значения по умолчанию»*. У пользователей всегда будет возникать желание настроить работу расширения под свои требования. Если вам удастся предугадать их желания и предоставить соответствующий механизм настройки, это увеличит шансы вашего расширения на более широкое использование. Но расширение должно быть как можно более простым в использовании, с минимумом обязательных настроек, поэтому все параметры должны иметь значения по умолчанию, подходящие для наиболее типичных случаев применения.

С этой целью:

- определяйте значения по умолчанию для всех параметров;
- упростите возможность локализации расширения, предоставляя отдельные параметры;
- реализуйте возможность чтения и изменения настроек;
- применяйте новые настройки немедленно;
- реализуйте возможность активации (enabling) и деактивации (disabling) расширения.

Рассмотрим каждый из этих пунктов по очереди.

5.5.1. Значения настроек по умолчанию

Вы уже видели, как сохранять значения по умолчанию в объекте-одиночке. Эти значения могут переопределяться пользователем при инициализации расширения, применяя его к своим элементам. Региональные настройки служат для локализации и более подробно описываются в следующем разделе. В листинге 5.8 перечислены параметры настройки для расширения MaxLength.

Листинг 5.8. Значения по умолчанию параметров настройки расширения

```
/* Диспетчер расширения MaxLength. */
function MaxLength() {
    this.regional = [];   // Доступные региональные настройки,
                          // индексируется кодом языка
    this.regional[''] = { // Региональные настройки по умолчанию
        feedbackText: '{r} characters remaining ({m} maximum)',
            // Отображаемый текст подсказки,
            // используйте {r} для вывода числа оставшихся (remaining)
            // символов, {c} - для вывода числа уже введенных символов,
            // {m} - для вывода максимально допустимого значения
        overflowText: '{o} characters too many ({m} maximum)'
            // Текст, отображаемый по достижении максимума,
            // используйте символы подстановки, описанные выше,
            // а также {o} - для вывода числа лишних символов
```

```
    };
    this._defaults = {
        max: 200,              // Максимальная длина
        truncate: true, // true: запретить ввод сверх ограничения,
            // false: только сообщить о превышении
        showFeedback: true,   // true: всегда отображать подсказку,
            // 'active' - отображать только при наведении указателя мыши
            // или при получении фокуса ввода
        feedbackTarget: null, // селектор jQuery или функция
            // для получения элемента, где должна выводиться подсказка
        onFull: null          // Функция, которая вызывается
            // при заполнении, или переполнении; принимает один параметр:
            // true, если зафиксировано переполнение,
            // false - в противном случае
    };
    $.extend(this._defaults, this.regional['']);
}
```

Возможность глобального изменения настроек для всех экземпляров расширения реализована в виде функции `setDefaults`, представленной в листинге 5.9. Настройки, передаваемые этой функции, добавляются в список значений по умолчанию и затем применяются ко всем вновь создаваемым экземплярам расширения, как было показано в разделе 5.4.2. Функция возвращает ссылку на объект-одиночку, что, в общем-то, не имеет большой практической ценности, потому что в нем нет других глобально доступных функций, но это соответствует практике поддержки цепочек вызовов в jQuery и никак не вредит работе в целом.

Листинг 5.9. Переопределение глобальных значений по умолчанию

```
/* Переопределяет настройки по умолчанию для всех экземпляров maxlength.
   @param  options (object) новые настройки по умолчанию
   @return (MaxLength) объект this */
setDefaults: function(options) {
    $.extend(this._defaults, options || {});
    return this;
},
```

Эта функция должна вызываться перед применением расширения к элементам, как показано ниже:

```
$.maxlength.setDefaults({max: 300, truncate: false});
```

5.5.2. Локализация

Чтобы расширение было доступно как можно более широкой аудитории, необходимо учитывать национальные особенности пользователей (реализовать принцип *добавления поддержки локализации*).

Наиболее очевидной национальной особенностью является язык (далеко не все говорят на английском языке), но национальные особенности также распространяются на форматы представления дат, чисел, денежных сумм и даже на направление письма – слева направо или справа налево. Вы можете существенно упростить локализацию расширения для ваших пользователей, сгруппировав соответствующие параметры.

Внутри объекта-одиночки определяется массив с параметрами локализации, который индексируется кодом языка (и, при необходимости, кодом региона). Язык по умолчанию (английский) определяется пустой строкой, а соответствующие ему настройки автоматически добавляются в расширение.

Пользователи, желающие локализовать расширение, должны определить новое множество значений параметров в настройках для своего региона (в массиве `regional`) и добавить их в настройки по умолчанию для всех экземпляров расширения.

Например, в листинге 5.10 приводится французская локализация для расширения. Настройки локализации должны храниться в файле с именем, соответствующим имени расширения, и с расширением, соответствующим языку. В данном случае `jquery.maxlength-fr.js`.

Листинг 5.10. Французская локализация для расширения MaxLength

```
/* http://keith-wood.name/maxlength.html
   Французская локализация для расширения MaxLength
   Автор: Кит Вуд (Keith Wood) (kbwood{at}iinet.com.au) апрель 2012. */
// ❶ Инкапсулировать внутренний код
(function($) { // скрытая область видимости

// ❷ Объявление локализованных значений
$.maxlength.regional['fr'] = {
    feedbackText: '{r} de caractères restants ({m} maximum)',
    overflowText: '{o} de caractères trop ({m} maximum)'
};
// ❸ Применение как параметров по умолчанию
//    ко всем экземплярам расширения
$.maxlength.setDefaults($.maxlength.regional['fr']);

})(jQuery);
```

Как и в случае с главной функцией расширения, для определения локализации создается новая область видимости, чтобы гарантировать, что идентификатор `$` будет ссылаться на объект `jQuery`, и чтобы скрыть особенности реализации ❶. Затем в массиве `regional` расширения создается новая запись с индексом в виде кода языка ❷.

Переопределяя параметры по умолчанию новыми локализованными настройками, вы упрощаете использование новой локализации ❸. Пользователю остается только подключить код расширения с дополнительной локализацией, использовать расширение в режиме по умолчанию и получить локализованное отображение на экране, как изображено на рис. 5.2. В листинге 5.11 показано, как загрузить и использовать французскую локализацию.

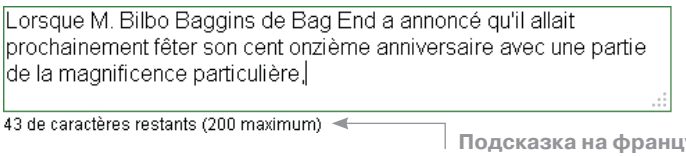


Рис. 5.2 ❖ Локализованное расширение MaxLength отображает информацию на французском языке

Листинг 5.11. Загрузка локализации

```
<script type="text/javascript" src="js/jquery.maxlength.js"></script>
<script type="text/javascript" src="js/jquery.maxlength-fr.js"></script>
<script type="text/javascript">
$(function() {
    $('#text1').maxlength(); // Французский язык будет выбран автоматически
});
</script>
```

Если потребуется задействовать некоторую конкретную локализацию уже после загрузки одной или нескольких других, можно просто сослаться на массив `regional` расширения и использовать его как новый параметр:

```
$('#text').maxlength($.maxlength.regional['fr']);
```

Или объединить с другими настройками:

```
$('#text').maxlength($.extend({max: 400}, $.maxlength.regional['fr']));
```

5.5.3. Реакция на изменение параметров

Как уже говорилось выше, вы должны *предусматривать возможность настройки расширения*, чтобы упростить его использование другими. Параметры настройки могут устанавливаться на этапе инициализации расширения, а также изменяться в любой момент на протяжении его жизненного цикла. Настройки могут влиять на внешний вид и/или поведение расширения, поэтому они должны применяться

немедленно. Кроме того, иногда бывает желательно иметь возможность получить значения текущих настроек. Все эти возможности предоставляются методом option.

В режиме чтения метод option позволяет указать имя интересующего параметра и возвращает его текущее значение (с учетом значений по умолчанию). Если имя параметра не указывается, возвращается полный набор параметров.

```
var maxChars = $('#text1').maxlength('option', 'max');
var options = $('#text1').maxlength('option');
```

В режиме записи метод option может изменить значение единственного параметра, если передается его имя и значение, или группы параметров, если передается объект.

```
$('#text1').maxlength('option', 'max', 400);
$('#text1').maxlength('option', {max: 400, truncate: false});
```

Конкретный режим работы определяется количеством и типами параметров, как показано в листинге 5.12.

Листинг 5.12. Чтение и запись значений параметров настройки

```
/* Извлекает или изменяет параметры настройки.
   @param  target  (element) элемент DOM
   @param  options (object)  новые параметры для данного экземпляра или
                   (string)  имя отдельного параметра
   @param  value   (любой)   значение отдельного параметра
                             (опускается, если options является объектом
                              или метод вызывается для чтения параметра)
   @return (любой), если метод вызван для чтения параметра */
// ❶ Определение метода option
_optionPlugin: function(target, options, value) {
    target = $(target);
    // ❷ Извлечь текущий экземпляр объекта с параметрами настройки
    var inst = target.data(this.propertyName);
    // ❸ Извлечь значение параметра
    if (!options || (typeof options == 'string' && value == null)) {
        // Получить параметр
        var name = options;
        options = (inst || {}).options;
        return (options && name ? options[name] : options);
    }
    // ❹ Проверить, было ли инициализировано расширение
    if (!target.hasClass(this.markerClassName)) {
        return;
    }
    options = options || {};
    // ❺ Изменить единственный параметр
```

```
    if (typeof options == 'string') {
        var name = options;
        options = {};
        options[name] = value;
    }
    // ❻ Сохранить новые значения параметров
    $.extend(inst.options, options);
    // Далее следует код, реализующий реакцию расширения
    // на изменение параметров
},
```

В листинге 5.12 определяется функция, реализующая метод `option` ❶. Имя функции следует соглашениям об именовании, установленным в разделе 5.4.3, – имя метода автоматически отображается в имя функции путем добавления к имени метода префикса из символа подчеркивания (_) и окончания `Plugin`.

Внутри функции извлекается объект с текущими настройками целевого элемента (`textarea`) ❷. Затем проверяется режим вызова ❸. Если пользователь не передал дополнительных параметров или передал только имя единственного параметра, вызывающей программе возвращаются либо все параметры настройки экземпляра, либо единственное значение, соответствующее параметру с указанным именем.

В противном случае считается, что метод `option` используется в режиме записи. Поэтому далее, как и в других функциях расширения, проверяется, было ли инициализировано расширение для данного элемента `textarea` ❹, и управление немедленно возвращается, если расширение не инициализировано. Если метод вызван, чтобы изменить значение единственного параметра ❺, имя параметра преобразуется в объект, чтобы избежать дублирования кода, выполняющего слияние.

Затем новые параметры сохраняются в текущем экземпляре ❻. Так как параметры могут оказывать влияние на внешний вид и/или поведение расширения, новые значения необходимо немедленно применить к целевому элементу.

Код, представленный выше, может использоваться во всех расширениях коллекций, где требуется сохранять некоторую информацию в элементах. Когда изменяется значение одного или нескольких параметров, необходимо сразу же применить эти изменения к текущему элементу, а реализация этой операции уже зависит от конкретного расширения. Тонкости реализации для расширения MaxLength обсуждаются в следующем разделе.

5.5.4. Реализация реакции на изменение параметров в MaxLength

Параметры, поддерживаемые расширением MaxLength, влияют на его внешний вид и/или поведение. При изменении параметров необходимо немедленно применить их к соответствующим элементам. Код, реализующий такое применение в данном расширении, представлен в листинге 5.13 и должен находиться в конце функции `_optionPlugin` из листинга 5.12 в разделе 5.5.3, после завершения стандартной обработки параметров.

Листинг 5.13. Применение новых значений параметров в расширении MaxLength

```
// ❶ Удалить прежний элемент с текстом подсказки
if (inst.feedbackTarget.length > 0) {
    // Удалить старый элемент
    if (inst.hadFeedbackTarget) {
        inst.feedbackTarget.empty().val('').
        removeClass(this._feedbackClass + ' ' +
            this._fullClass + ' ' + this._overflowClass);
    }
    else {
        inst.feedbackTarget.remove();
    }
    inst.feedbackTarget = $([]);
}
// ❷ Добавить новый элемент с текстом подсказки
if (inst.options.showFeedback) {
    // Добавить новый элемент
    inst.hadFeedbackTarget = !!inst.options.feedbackTarget;
    if ($.isFunction(inst.options.feedbackTarget)) {
        inst.feedbackTarget =
            inst.options.feedbackTarget.apply(target[0], []);
    }
    else if (inst.options.feedbackTarget) {
        inst.feedbackTarget = $(inst.options.feedbackTarget);
    }
    else {
        inst.feedbackTarget = $('<span></span>').insertAfter(target);
    }
    inst.feedbackTarget.addClass(this._feedbackClass);
}
// ❸ Выводить подсказку, только когда элемент активен
target.unbind('mouseover.maxlength focus.maxlength ' +
    'mouseout.maxlength blur.maxlength');
if (inst.options.showFeedback == 'active') {
    // Дополнительные обработчики событий
```

```
    target.bind('mouseover.maxlength', function() {
            inst.feedbackTarget.css('visibility', 'visible');
        }).bind('mouseout.maxlength', function() {
            if (!inst.focussed) {
                inst.feedbackTarget.css('visibility', 'hidden');
            }
        }).bind('focus.maxlength', function() {
            inst.focussed = true;
            inst.feedbackTarget.css('visibility', 'visible');
        }).bind('blur.maxlength', function() {
            inst.focussed = false;
            inst.feedbackTarget.css('visibility', 'hidden');
        });
    inst.feedbackTarget.css('visibility', 'hidden');
}
// ❹ Выполнить проверку текущей длины текста в поле ввода
this._checkLength(target);
```

Расширение MaxLength позволяет отображать информационную подсказку либо в самом целевом элементе, управляемом расширением, либо в другом элементе, указанном пользователем. Если выводить подсказку больше не требуется или пользователь определил внешний элемент для вывода подсказки, необходимо удалить предыдущую подсказку ❶. Затем, с учетом новых настроек, настраивается новый элемент для вывода подсказки ❷.

Действие конструкции `!!` описывается в разделе 3.2.1.

Аналогично удаляются все обработчики событий, отвечавшие за отображение подсказки при активации элемента `textarea`, и добавляются вновь, если это необходимо ❸.

Обратите внимание, что идентификаторы событий включают пространства имен. Благодаря этому ваши действия не скажутся на обработчиках событий, установленных другими расширениями.

В заключение вызывается функция `_checkLength` ❹, которая реализует основную цель расширения – применение ограничения длины содержимого в элементе `textarea`.

5.5.5. Активация и деактивация виджета

Несмотря на отсутствие соответствующего параметра настройки, возможность включения и отключения функциональных возможностей расширений часто используется на практике и влияет на их внешний вид и поведение. В данной инфраструктуре изменение состояния расширения выполняется вызовом методов `enable/disable`.

```
$('#text1').maxlength('disable');
...
$('#text1').maxlength('enable');
```

То есть нам нужны соответствующие функции _enablePlugin и _disablePlugin, реализующие данные методы.

Листинг 5.14. Включение и выключение расширения

```
/* Активирует элемент DOM.
   @param  target  (element)  целевой элемент DOM */
// ❶ Определение функций активации/деактивации
_enablePlugin: function(target) {
    target = $(target);
    // ❷ Проверить, инициализировано ли расширение
    if (!target.hasClass(this.markerClassName)) {
        return;
    }
    // ❸ Активировать/деактивировать целевой элемент DOM
    target.prop('disabled', false).removeClass('maxlength-disabled');
    var inst = target.data(this.propertyName);
    inst.feedbackTarget.removeClass('maxlength-disabled');
},

/* Деактивирует элемент DOM.
   @param  target  (element) целевой элемент DOM */
// ❶ Определение функций активации/деактивации
_disablePlugin: function(target) {
    target = $(target);
    // ❷ Проверить, инициализировано ли расширение
    if (!target.hasClass(this.markerClassName)) {
        return;
    }
    // ❸ Активировать/деактивировать целевой элемент DOM
    target.prop('disabled', true).addClass('maxlength-disabled');
    var inst = target.data(this.propertyName);
    inst.feedbackTarget.addClass('maxlength-disabled');
},
```

В листинге 5.14 определяются функции, реализующие соответствующие им методы. Имена функций следуют принятым соглашениям об именовании – имя метода автоматически отображается в имя функции путем добавления префикса из символа подчеркивания (_) и окончания Plugin ❶. Управление этим функциям передается при обращении к соответствующим методам, как показано в разделе 5.4.3.

Каждая функция принимает единственный параметр – ссылку на целевой элемент textarea. В начале каждой функции проверяется, было ли инициализировано расширение MaxLength для данного эле-

мента textarea ❷, и управление немедленно возвращается вызывающей программе, если расширение не инициализировано. В противном случае сбрасывается или устанавливается атрибут disabled ❸ и удаляются или устанавливаются соответствующие классы, применяемые для нужд визуального оформления самого поля ввода и элемента вывода подсказки. Никаких дополнительных действий выполнять не требуется, так как деактивированный элемент textarea сам позаботится о соответствующей реакции на нажатия клавиш.

5.6. Добавление обработчиков событий

Возможность подключения собственных обработчиков событий позволяет пользователю реализовать собственную реакцию на важные события, происходящие внутри расширения. JavaScript поддерживает передачу ссылок на функции подобно простым строкам и числам и предоставляет механизм для вызова этих функций с соответствующим контекстом и параметрами.

Чтобы пользователь смог добавлять свои обработчики событий, необходимо:

- позволить ему регистрировать обработчики;
- реализовать возбуждение событий в соответствующие моменты времени.

5.6.1. Регистрация обработчиков событий

Расширение MaxLength предоставляет только одно событие, которое может обрабатываться пользователем, – onFull. Оно возбуждается, когда длина содержимого элемента textarea достигает установленного максимума. Так как другие параметры настройки могут разрешать ввод символов сверх установленного ограничения (truncate: false), вы должны также сообщить пользователю, что длина содержимого textarea достигла предела, но остается допустимой, или превысила его и текст требуется сократить перед отправкой на сервер.

Обработчик события onFull оформлен как еще один параметр настройки (листинг 5.15) со значением по умолчанию null, указывающим на отсутствие обработчика. При необходимости этому параметру должна быть присвоена ссылка на функцию, принимающую единственный параметр – флаг переполнения элемента textarea. Внутри этого обработчика переменная this будет ссылаться на сам элемент textarea, что дает пользователю возможность обрабатывать события от нескольких элементов с помощью одного и того же обработчика.

Листинг 5.15. Определение обработчика событий

```
/* Диспетчер расширения MaxLength. */
function MaxLength() {
    this.regional = [];   // Доступные региональные настройки,
                          // индексируется кодом языка
    this.regional[''] = { // Региональные настройки по умолчанию
        ...
    };
    this._defaults = {
        ...
        onFull: null // Обработчик событий заполнения и переполнения,
                     // принимает один параметр:
                     // true - если переполнение,
                     // false - просто достигнуто ограничение
    };
    $.extend(this._defaults, this.regional['']);
}
```

Регистрация обработчика выполняется на этапе инициализации расширения, как показано ниже, или позднее, посредством изменения параметров:

```
$('#text1').maxlength({onFull: function(overflow) {
    $('#warning').html(overflow ? 'Overflowed' : 'Full').show();
}});
```

5.6.2. Вызов обработчика события

Обработчик события вызывается в конце функции _checkLength, после применения ограничения на длину текста, содержащегося в элементе ввода, как показано в листинге 5.16. В других расширениях обработчики вызываются в соответствующих функциях.

Листинг 5.16. Вызов обработчика события

```
/* Проверяет длину текста и при необходимости вызывает обработчик.
   @param  target  (jQuery) проверяемый элемент DOM */
_checkLength: function(target) {
    var inst = target.data(this.propertyName);
    var value = target.val();
    var len = value.replace(/\r\n/g, '~~').replace(/\n/g, '~~').length;
    ...
    // ❶ Проверить необходимость вызова и наличие обработчика
    if (len >= inst.options.max && $.isFunction(inst.options.onFull)) {
        // ❷ Вызвать обработчик
        inst.options.onFull.apply(target, [len > inst.options.max]);
    }
},
```

Для начала необходимо проверить необходимость вызова обработчика и наличие ссылки на функцию обратного вызова в параметрах настройки ❶ и только потом вызвать эту функцию ❷. Стандартная функция `apply` позволяет установить контекст для вызываемой функции и передать ей любые параметры. Свой первый параметр она присваивает переменной `this`, которая будет доступна в вызываемой функции.

5.7. Добавление методов

Многие функциональные возможности расширения реализованы в виде методов. Чтобы добавить новый метод, достаточно просто определить функцию с именем, соответствующим соглашениям, принятым в расширении. Если метод возвращает некоторое значение, отличное от текущего объекта коллекции `jQuery`, его необходимо дополнительно добавить в массив методов чтения, как показано в разделе 5.4.4.

5.7.1. Получение текущей длины

Расширение MaxLength позволяет извлекать текущее число символов в конкретном элементе. Метод `curLength`, представленный в листинге 5.17, возвращает объект с атрибутами: `used`, хранящим число введенных символов, и `remaining`, хранящим число символов, которые еще можно ввести. Обратите внимание, что значение атрибута `used` может быть больше параметра настройки `max`, а атрибут `remaining` может иметь отрицательное значение, если для элемента `textarea` разрешено вводить текст, длина которого превышает установленный предел.

Листинг 5.17. Извлечение текущей длины

```
/* Возвращает число символов, введенных и оставшихся до предельного зна-
чения.
   @param   target   (jQuery) целевой элемент DOM
   @return  (object) текущие счетчики символов в виде атрибутов
                     used и remaining */
// ❶ Функция, реализующая метод curLength
_curLengthPlugin: function(target) {
    var inst = target.data(this.propertyName);
    var value = target.val();
    var len = value.replace(/\r\n/g, '~~').replace(/\n/g, '~~').length;
    // ❷ Вернуть текущие значения счетчиков
    return {used: len, remaining: inst.options.max - len};
},
```

В соответствии с соглашениями имя функции конструируется из имени метода путем добавления префикса в виде символа подчеркивания (_) и стандартного окончания Plugin ❶, благодаря чему механизм вызова, представленный в разделе 5.4.4, сможет найти и вызвать эту функцию. Функция формирует значение, соответствующее целевому элементу ❷, и возвращает его непосредственно пользователю.

5.8. Удаление расширения

Расширения добавляют к элементам, найденным в странице, дополнительные функциональные возможности, способствующие улучшению впечатлений, получаемых конечным пользователем. Но иногда бывает необходимо удалить все дополнительные возможности и вернуть элементы в первоначальное состояние. Вызов метода destroy указывает расширению, что оно должно удалить все следы своего пребывания.

5.8.1. Метод destroy

Функция _destroyPlugin, представленная в листинге 5.18, реализует метод destroy. Как и другие методы, он вызывается обращением к главной функции расширения, описанной в разделе 5.4.3, и принимает единственный параметр – ссылку на целевой элемент textarea.

Листинг 5.18. Удаление функциональных возможностей расширения

```
/* Удаляет функциональные возможности расширения из элемента DOM.
   @param  target  (element) целевой элемент DOM */
// ❶ Определение функции, реализующей метод destroy
_destroyPlugin: function(target) {
    target = $(target);
    // ❷ Проверить, инициализировано ли расширение
    if (!target.hasClass(this.markerClassName)) {
        return;
    }
    var inst = target.data(this.propertyName);
    // ❸ Удалить любые элементы, использовавшиеся
    //   для вывода подсказки
    if (inst.feedbackTarget.length > 0) {
        if (inst.hadFeedbackTarget) {
            inst.feedbackTarget.empty().val('').
                css('visibility', 'visible').
                removeClass(this._feedbackClass + ' ' +
                    this._fullClass + ' ' + this._overflowClass);
        }
        else {
            inst.feedbackTarget.remove();
        }
```

```
    }
    // ❹ Удалить функциональные возможности расширения
    target.removeClass(this.markerClassName + ' ' +
            this._fullClass + ' ' + this._overflowClass).
        removeData(this.propertyName).
        unbind('.maxlength');
}
```

Как и в случае с другими методами, имя функции с реализацией метода `destroy` выбирается так, чтобы она могла быть вызвана при обращении к нему ❶. Функция проверяет, было ли инициализировано данное расширение, и управление немедленно возвращается вызывающей программе, если расширение не инициализировано ❷. В противном случае отменяются все действия, выполненные функциями `_attachPlugin` и `_optionPlugin`.

Сначала проверяется наличие элемента для вывода подсказки, и этот элемент либо восстанавливается в исходное состояние (если он был указан пользователем через параметр настройки), либо удаляется совсем (если был создан самим расширением) ❸. Затем из элемента `textarea` удаляются классы, установленные расширением, данные и обработчики событий ❹. Использование пространства имен при подключении обработчиков событий упрощает их удаление. Любые другие обработчики событий, подключенные к элементу `textarea`, останутся нетронутыми.

После выполнения этой функции страница вернется в исходное состояние и более не будет поддерживать возможности расширения MaxLength.

5.9. Заключительные штрихи

К настоящему моменту инфраструктура поддержки расширений представлена полностью. Большая часть кода инфраструктуры может быть повторно использована в других расширениях коллекций. Но нам осталось внести еще пару заключительных штрихов в само расширение:

- реализация главной цели расширения;
- поддержка стилей оформления, определяющих внешний вид.

5.9.1. Главная цель расширения

Главная цель расширения MaxLength – ограничить длину текста, который можно ввести в элемент `textarea`. В предыдущих разделах было показано, как реализовать главную функцию расширения, интегри-

рующегося с библиотекой jQuery, и как определить комплект методов в виде внутренних функций. При инициализации расширения или изменении его параметров настройки в конечном итоге вызывается функция _checkLength, выполняющая основную работу. В листинге 5.19 показано, как она действует.

Листинг 5.19. Ограничение длины текста в элементе textarea

```
/* Проверяет длину текста и выводит соответствующую подсказку.
   @param  target  (jQuery) целевой элемент DOM */
_checkLength: function(target) {
    var inst = target.data(this.propertyName);
    var value = target.val();
    // ❶ Нормализовать окончания строк
    var len = value.replace(/\r\n/g, '~~').replace(/\n/g, '~~').length;
    // ❷ Установить текущее состояние элемента textarea
    target.toggleClass(this._fullClass, len >= inst.options.max).
        toggleClass(this._overflowClass, len > inst.options.max);
    // ❸ Выполнить усечение текста
    if (len > inst.options.max && inst.options.truncate) {
        // Усечение
        var lines = target.val().split(/\r\n|\n/);
        value = '';
        var i = 0;
        while (value.length < inst.options.max && i < lines.length) {
            value += lines[i].substring(
                0, inst.options.max - value.length) + '\r\n';
            i++;
        }
        target.val(value.substring(0, inst.options.max));
        // Прокрутить вниз
        target[0].scrollTop = target[0].scrollHeight;
        len = inst.options.max;
    }
    // ❹ Установить текущее состояние подсказки
    inst.feedbackTarget.
        toggleClass(this._fullClass, len >= inst.options.max).
        toggleClass(this._overflowClass, len > inst.options.max);
    // ❺ Сгенерировать и вывести текст подсказки
    var feedback = (len > inst.options.max ? // Подсказка
        inst.options.overflowText : inst.options.feedbackText).
            replace(/\{c\}/, len).
            replace(/\{m\}/, inst.options.max).
            replace(/\{r\}/, inst.options.max - len).
            replace(/\{o\}/, len - inst.options.max);
    try {
        inst.feedbackTarget.text(feedback);
    }
    catch(e) {
```

```
            // Игнорировать
        }
        try {
            inst.feedbackTarget.val(feedback);
        }
        catch(e) {
            // Игнорировать
        }
        // ❻ Вызвать обработчик события, если необходимо
        if (len >= inst.options.max && $.isFunction(inst.options.onFull)) {
            inst.options.onFull.apply(target, [len > inst.options.max]);
        }
},
```

После извлечения информации об экземпляре расширения для данного элемента `textarea` функция определяет текущую длину текста, учитывая различия между браузерами в оформлении окончаний строк ❶. Опираясь на это значение и параметры настройки расширения, к элементу `textarea` применяется один или два класса, чтобы отразить его текущее состояние – заполнен или переполнен ❷.

Если длина текста превышает установленный максимум и пользователь потребовал усекать лишний текст ❸, вычисляется усеченное значение, при этом снова выполняется нормализация окончаний строк. Когда текст будет записан обратно в элемент `textarea`, он автоматически прокрутит содержимое вверх. Однако обычно текст вводится в конце, поэтому далее выполняется прокрутка вниз, чтобы помочь пользователю. Так как было произведено усечение текста, текст подсказки должен измениться, чтобы отразить этот факт.

Далее, исходя из новой длины текста, к элементу подсказки применяется один или два класса ❹. Так как длина текста может уменьшиться из-за его усечения, между элементами `textarea` и подсказки могут появиться расхождения. Это сделано преднамеренно, поскольку подсказка должна отражать обновленное состояние, а элемент `textarea` своим внешним видом должен показывать, что предпринята неудачная попытка ввести дополнительный текст.

Состояние элемента `textarea` отображается в любом элементе подсказки в виде одного из двух сообщений, выбор которого зависит от состояния переполнения ❺. Для большей гибкости в качестве элемента подсказки расширение допускает использовать `div`, `span`, `p` или `input`. Но текст в эти элементы записывается по-разному, и использование неправильного способа может вызвать исключение в некоторых браузерах. Поэтому запись текста подсказки выполняется здесь внутри двух инструкций `try/catch`.

Наконец, чтобы сообщить о наполнении или переполнении элемента, вызывается обработчик `onFull` ❻. Тема вызова обработчика события подробно рассматривалась в разделе 5.6.2.

5.9.2. Реализация поддержки стилей

Теперь в вашем распоряжении имеется полноценное расширение для jQuery, реализующее дополнительные функциональные возможности. Но, возможно, оно имеет не самый привлекательный внешний вид. Вам следует добавить в комплект расширения дополнительный файл CSS и настроить его внешний вид в соответствии с принципом, требующим *оформлять внешний вид с помощью CSS*. Имя файла должно совпадать с именем файла, где хранится код расширения, но расширение должно быть другим. В данном случае оформление должно быть определено в файле `jquery.maxlength.css`. Использование таблиц стилей CSS вместо параметров настройки внутри самого расширения дает пользователю возможность переопределять и настраивать внешний вид расширения с минимальными усилиями.

В листинге 5.20 представлено содержимое файла CSS для расширения MaxLength. В нем определяются классы, определяющие внешний вид элементов, которые создаются и управляются расширением. Например, ошибочное состояние подчеркивается классами `maxlength-full` и `maxlength-overflow`, которые изменяют цвет фона поля ввода.

Листинг 5.20. Стили CSS для расширения MaxLength

```
/* Стили CSS для расширения MaxLength v2.0.0 */
.maxlength-feedback {
    margin-left: 0.5em;
    font-size: 75%;
}
.maxlength-full {
    background-color: #fee;
}
.maxlength-overflow {
    background-color: #fcc;
}
.maxlength-disabled {
    opacity: 0.5;
}
```

Опираясь на классы, присвоенные элементам, находящимся под управлением расширения, механизм поддержки CSS генерирует желаемый внешний вид. Всего несколькими строчками кода CSS пользователь может изменить его внешний вид, как показано на рис. 5.3.

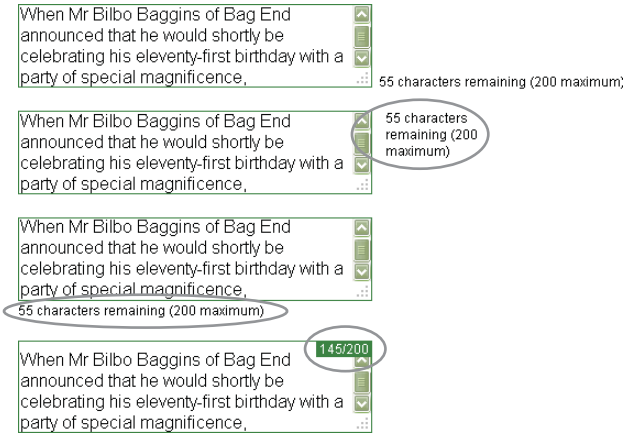


Рис. 5.3 ❖ Оформление расширения MaxLength: стиль по умолчанию, компактный стиль, подсказка под элементом ввода и подсказка поверх элемента ввода

5.10. Законченное расширение

Итак, мы закончили разработку расширения, ограничивающего длину текста в поле ввода `textarea`. Вы видели, как действует моя инфраструктура поддержки расширений и как она реализует принципы проектирования, описанные в главе 4. Полный код расширения доступен для загрузки на сайте книги.

> На сайте можно также найти ужатый файл (`jquery.framework.js`), содержащий только саму инфраструктуру. Вы можете использовать его как основу для своих будущих расширений.

Чтобы задействовать новое расширение в веб-странице, следует сначала загрузить библиотеку jQuery, затем расширение и стили и наконец подключить расширение к выбранным элементам. В листинге 5.21 приводится простая веб-страница, демонстрирующая описанные действия.

Листинг 5.21. Использование расширения MaxLength

```
<html>
<head>
<title>jQuery Max Length Basics</title>
<!-- ❶ Загрузить стили для расширения -->
<link type="text/css" href="jquery.maxlength.css" rel="stylesheet">
```

```
<!-- ❷ Загрузить библиотеку jQuery -->
<script type="text/javascript"
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js">
</script>
<!-- ❸ Загрузить расширение -->
<script type="text/javascript" src="jquery.maxlength.js"></script>
<script type="text/javascript">
$(function() {
    // ❹ Подключить расширение к элементу
    $('#maxLength').maxlength();
});
</script>
</head>
<body>
<h1>jQuery Max Length Basics</h1>
<p>This page demonstrates the very basics of the
    <a href="http://keith-wood.name/maxlength.html">jQuery
    Max Length plugin</a>.
    It contains the minimum requirements for using the plugin and
    can be used as the basis for your own experimentation.</p>
<p>For more detail see the
    <a href="http://keith-wood.name/maxlengthRef.html">
    documentation reference</a> page.</p>
<p><span class="demoLabel">Default max length:</span>
    <!-- ❺ Целевой элемент -->
    <textarea id="maxLength" rows="5" cols="50"></textarea></p>
</body>
</html>
```

В разделе `head` документа HTML выполняется загрузка стилей расширения MaxLength ❶, библиотеки jQuery ❷ (часто из сети доставки содержимого CDN, как показано здесь) и самого расширения MaxLength ❸. Внутри функции обратного вызова `document.ready` (чтобы гарантировать, что дерево DOM загружено полностью, прежде чем обращаться к нему) расширение подключается к целевому элементу ❹ поля ввода. Само поле ввода находится в теле документа ❺.

Это нужно знать

Создавайте расширения коллекций для выполнения операций с множествами элементов, полученными в результате применения селекторов или непосредственного обхода дерева DOM.

Использование описанной здесь инфраструктуры в качестве основы позволяет сосредоточиться исключительно на реализации функциональных возможностей расширения.

Подключайте расширения коллекций к точке интеграции `$.fn`.

Защищайте свои расширения и предотвращайте конфликты имен с помощью создания областей видимости.

Обеспечивайте поддержку составления цепочек вызовов с другими функциями jQuery.

Предоставляйте параметры настройки в своих расширениях, но всегда инициализируйте их осмысленными значениями по умолчанию.
Используйте именованные методы для предоставления дополнительных функциональных возможностей.
Позволяйте удалять расширение, реализуя метод `destroy`.

Попробуйте сами

Используя файл фреймворка (`jquery.framework.js`) в качестве основы, реализуйте еще раз расширение Watermark из главы 2. Добавьте параметр для идентификации атрибута, откуда будет извлекаться текст для метки. Вместо отдельного расширения добавьте метод, который будет выполнять очистку метки.

5.11. В заключение

Расширения коллекций оперируют множествами элементов, полученными в результате применения селекторов или непосредственного обхода дерева DOM. Такие расширения добавляются к точке интеграции `$.fn` с целью определить новые функции и интегрировать их в стандартный механизм библиотеки jQuery.

Пример расширения, построенного в этой главе, иллюстрирует, как инфраструктура поддержки расширений реализует различные принципы проектирования, описанные в главе 4. Как результат теперь у вас имеется законченное расширение, предоставляющее дополнительные возможности, которые можно встраивать в обычные поля ввода.

В следующей главе мы создадим расширение другого типа, которое не оперирует коллекциями элементов.

Глава 6

Расширения-функции

Эта глава охватывает следующие темы:

- определение расширений-функций;
- локализация содержимого с помощью расширений-функций;
- доступ к cookies с помощью расширения-функции.

Расширения коллекций, о которых рассказывалось в предыдущей главе, оперируют множествами элементов страниц, полученными в результате применения селекторов или непосредственного обхода дерева DOM. Но вы также имеете возможность создавать расширения, которые не применяются к коллекциям, а предоставляют вспомогательные функции. Такие расширения называют *расширениями функций*.

Примерами расширений этого типа могут служить: расширение Debug (http://jquery.glyphix.com/), обеспечивающее поддержку вывода отладочных сообщений, и Cookie, позволяющее обрабатывать cookies веб-сайта (подробно рассматривается в разделе 6.2). Широта возможностей расширений функций, как и расширений коллекций, ограничивается только вашей фантазией.

Так как расширения-функции не предназначены для работы с коллекциями элементов и часто вообще не имеют никакого отношения к компонентам пользовательского интерфейса, они существенно проще в реализации. Несмотря на то что подобные расширения можно оформить в виде самостоятельных функций JavaScript, включение их в пространство имен jQuery имеет свои преимущества. При таком подходе уменьшается загрязнение глобального пространства имен и снижается риск конфликтов с другими библиотеками. Часто такие функции сами используют jQuery, и их включение в библиотеку обеспечивает единообразие применения. Они также имеют своей целью упростить их использование и скрыть любые различия между браузерами: ключевые принципы, лежащие в основе jQuery.

В главе 5 мы подключали свое расширение коллекций к точке интеграции `$.fn`, чтобы интегрировать его в механизм обработки кол-

лекций jQuery. Расширения-функции подключаются непосредственно к объекту `jQuery` (`$`) и вызываются относительно него.

6.1. Определение расширения

Как конкретный пример расширения-функции мы создадим инструмент, помогающий локализовать веб-страницы, загружая только необходимые файлы JavaScript для настройки сайта на использование определенного языка.

6.1.1. Расширение для локализации

Основанный на схеме локализации, описанной в разделе 5.5.2, этот инструмент предполагает наличие различных файлов JavaScript, отличающихся только кодами языка и региона в их именах. Расширение загружает файлы локализации в порядке увеличения специфичности языка и региона, затирая прежде загруженный файл следующим, что в результате дает наилучшее соответствие для данного языка и региона.

Например, допустим, что имеются файлы, перечисленные в табл. 6.1. Каждый файл соответствует определенной комбинации языка и региона и присваивает общей переменной (`greeting`) текст сообщения для последующего отображения.

Таблица 6.1. Файлы локализации

Файл	Язык/регион	Текст сообщения
greeting.js	По умолчанию	Hello
greeting-en.js	Английский	Good day
greeting-en-US.js	Английский (США)	Hi
greeting-en-AU.js	Английский (Австралия)	G'day
greeting-fr.js	Французский	Bonjour

Для локализации текста приветствия в веб-странице можно использовать следующий код:

```
$.localise('greeting', 'en-AU');
$('#greet').text(greeting);
```

Расширение должно в этом случае загрузить файлы в порядке от менее специфичных к более специфичным – `greeting.js`, `greeting-en.js`, `greeting-en-AU.js`, чтобы получить наилучшее соответствие для австралийского английского: «G'day». Если, к примеру, будет запрошена локализация для канадского английского (en-CA), загрузка

остановится на втором файле (стандартный английский), потому что файл `greeting-en-CA.js` отсутствует, и на странице появится текст «Good day». А если запросить текст на языке кхоса (xh), будет выведен текст приветствия по умолчанию: «Hello».

Как только файлы будут загружены и выполнены, они установят переменные самым обычным образом.

Если функцию `localise()` вызвать без кода языка, будет использоваться язык по умолчанию, определяемый настройками браузера. Чтобы упростить получение кода языка по умолчанию, он сохраняется в атрибуте `$.localise.defaultLanguage`.

Локализация не ограничивается переводом текстовых строк на язык конечного пользователя. Она может влиять и на другие аспекты отображения информации, как показано на рис. 6.1, где демонстрируются варианты отображения локализованных версий расширения Datepicker.

Рис. 6.1 ❖ Локализация расширения Datepicker: французский, японский и арабский языки

Между этими версиями имеется несколько различий, кроме очевидных изменений в текстовых значениях:

- раскрывающиеся списки выбора месяца и года располагаются в разном порядке;
- в каждом из календарей неделя начинается с разных дней: во французской версии – с понедельника, в японской – с воскресенья, в арабской – с субботы;
- в арабской версии используется письмо справа налево, в других версиях – слева направо.

Теперь, когда вы знаете, как проектируются расширения, вы сможете реализовать все это, используя принципы и инфраструктуру из двух предыдущих глав.

6.1.2. Код инфраструктуры

Несмотря на то что большая часть инфраструктуры поддержки расширений, представленной в предыдущей главе, непригодна для создания расширений функций, пара ее особенностей все же сохранятся и в данной инфраструктуре. В расширениях функций все еще желательно *скрывать тонкости реализации с использованием областей видимости* и *не полагаться, что имя $ будет ссылаться на jQuery*. Реализация этих двух принципов выглядит точно так же, как показано в листинге 6.1.

Листинг 6.1. Инкапсуляция реализации расширения

```
// ❶ Объявление анонимной функции
(function($) { // Скрывает из видимости внешнего кода,
               // устраняет конфликты с идентификатором $
    ... здесь располагается остальной код
})(jQuery); // ❷ Немедленный вызов функции
```

Здесь создается анонимная функция, играющая роль новой области видимости ❶, и затем она немедленно вызывается ❷. Объявление параметра с именем `$` и передача в нем ссылки на объект `jQuery` при вызове обеспечивают возможность использования двух этих имен для ссылки на один и тот же объект в теле функции.

Расширение также следует принципам *создания единственного имени и использования только его для взаимодействий с расширением* и *размещения всего необходимого в объекте jQuery*. Расширения-функции не участвуют в цепочках вызовов, и им не приходится обрабатывать коллекции элементов. Следование принципу *использования осмысленных значений по умолчанию* обеспечивается использованием региональных настроек в браузере, если код языка не задан в вызове функции.

Все эти вопросы подробно рассматриваются в следующих разделах, где демонстрируется процесс реализации главной функции расширения.

6.1.3. Загрузка локализаций

Функция `localise`, представленная в листинге 6.2, реализует главную особенность расширения: она загружает один или более файлов локализации для данного пакета, опираясь на код языка и региона.

Листинг 6.2. Объявление функции расширения

```
// ❶ Объявление функции localize
$.localise = function(
```

```
        packages, language, loadBase, path,
        timeout, async, complete) {
    ...
};
```

Функция подключается непосредственно к объекту `jQuery` путем объявления ее как атрибута ❶. После этого она становится доступна внутри страницы через объект `jQuery` и не требует выбирать коллекцию элементов, потому что применяется к странице в целом.

Функция имеет множество параметров, оказывающих влияние на ее поведение. Все параметры, кроме первого, являются необязательными и получают соответствующие значения по умолчанию, если они не указываются явно. Назначение параметров определяется их типами. В листинге 6.3 показано, как производится обработка параметров.

Листинг 6.3. Стандартизация значений параметров и присваивание им значений по умолчанию

```
// ❶ Обработка необязательного кода языка
if (typeof language != 'object' && typeof language != 'string') {
    complete = async;
    async = timeout;
    timeout = path;
    path = loadBase;
    loadBase = language;
    language = '';
}
// ❷ Обработка необязательного параметра, определяющего
//    необходимость повторной загрузки базовой локализации
if (typeof loadBase != 'boolean') {
    complete = async;
    async = timeout;
    timeout = path;
    path = loadBase;
    loadBase = false;
}
// ❸ Обработка необязательного параметра пути
if (typeof path != 'string' && !$.isArray(path)) {
    complete = async;
    async = timeout;
    timeout = path;
    path = '';
}
// ❹ Обработка необязательного параметра, определяющего тайм-аут
if (typeof timeout != 'number') {
    complete = async;
    async = timeout;
    timeout = 500;
```

```
}
// ❺ Обработка необязательного параметра,
//    определяющего асинхронный режим работы
if (typeof async != 'boolean') {
    complete = async;
    async = false;
}
// ❻ Привести параметры настройки к стандартному виду
var settings = (typeof language != 'string' ? $.extend(
    {loadBase: false, path: '', timeout: 500, async: false},
    language || {}) :
    {language: language, loadBase: loadBase, path: path,
    timeout: timeout, async: async, complete: complete});
// ❼ Привести настройки пути к стандартному виду
var paths = (!settings.path ? ['', ''] :
    ($.isArray(settings.path) ? settings.path :
    [settings.path, settings.path]));
// ❽ Подготовка параметров для последующего запроса Ajax
var opts = {async: settings.async, dataType: 'script',
    timeout: settings.timeout};
```

При вызове функции необходимо указать имя единственного пакета (`string`) или массив имен (`string[]`). Так как остальные параметры являются необязательными, реализация должна определить, какой параметр какое значение определяет. Второй параметр – необязательный код языка (`string`) или коллекция параметров (`object`) с атрибутами, имена которых соответствуют отдельным параметрам. Чтобы упростить дальнейшую обработку, любые отдельные параметры собираются в объект строк, как если бы этот объект был передан функции изначально. Если второй параметр не является значением одного из этих типов, производится сдвиг всех параметров ❶, и обработка продолжается со следующего параметра.

В число дополнительных, необязательных параметров входят: флаг, определяющий необходимость повторной загрузки базовой локализации (`boolean`) ❷; путь (`string`) или пути (`string[]`) к файлам локализации ❸; значение тайм-аута (`number`) ❹; признак асинхронного режима (`boolean`) ❺ и ссылка на функцию, которая должна быть вызвана по окончании загрузки. Отсутствующие параметры получают соответствующие значения по умолчанию.

После обработки отдельных параметров создается объект с параметрами настройки из объекта, переданного в вызов функции (указанные в нем параметры замещают значения по умолчанию), или из значений отдельных параметров ❻. Далее проверяется настройка пути, чтобы выяснить, был ли указан единственный путь или несколько, и выполняется преобразование в массив, если необходимо

❼. Наконец, производится объединение параметров, определяющих настройки для последующего запроса Ajax ❽.

После стандартизации всех параметров и присваивания значений по умолчанию, если это необходимо, расширение приступает к обработке всех указанных пакетов, как показано в листинге 6.4.

Листинг 6.4. Загрузка файлов локализации

```
// ❶ Лоализовать отдельный пакет
var localisePkg = function(pkg, lang) {
    var files = [];
    // ❷ Загрузить базовый файл, если необходимо
    if (settings.loadBase) {
        files.push(paths[0] + pkg + '.js');
    }
    // ❸ Загрузить локализацию для указанного языка
    if (lang.length >= 2) {
        files.push(paths[1] + pkg + '-' +
            lang.substring(0, 2) + '.js');
    }
    // ❹ Загрузить локализацию для указанного языка и региона
    if (lang.length >= 5) {
        files.push(paths[1] + pkg + '-' +
            lang.substring(0, 5) + '.js');
    }
    // ❺ Загрузить файлы по порядку
    var loadFile = function() {
        $.ajax($.extend(opts, {url: files.shift(),
        complete: function() {
            // ❻ Вызвать функцию после загрузки всех файлов
            if (files.length == 0) {
                if ($.isFunction(settings.complete)) {
                    settings.complete.apply(window, [pkg]);
                }
            }
            else {
                // ❼ Загрузить следующий файл
                loadFile();
            }
        }}));
    }
    loadFile();
};
// ❽ Привести код языка к стандартному виду
//    для последующего использования
var lang = normaliseLang(
    settings.language || $.localise.defaultLanguage);
// ❾ Привести список пакетов к стандартному виду
//    и обработать их
packages = ($.isArray(packages) ? packages : [packages]);
```

```
for (var i = 0; i < packages.length; i++) {
    localisePkg(packages[i], lang);
}
```

Здесь объявляется внутренняя функция ❶ для обработки единственного пакета. Каждый пакет определяется как коллекция файлов: сначала базовый файл (если установлен соответствующий флаг в параметрах настройки) ❷, затем файл с двухсимвольным кодом языка в имени ❸ и, наконец, файл с пятисимвольным кодом языка и региона (если доступны) ❹. Файлы помещаются в очередь, откуда будут извлекаться во время загрузки последовательно, друг за другом.

Загрузка отдельных файлов ❺ осуществляется с помощью функции `ajax` из библиотеки jQuery и начинается с первого файла в очереди. При этом в одном из параметров настройки функции `ajax` указывается, что содержимым каждого файла является «сценарий» и потому его необходимо выполнить сразу после загрузки. Так как по умолчанию используется синхронный режим загрузки (чтобы последующий код мог опираться на возвращаемое значение), для каждого файла устанавливается тайм-аут. При желании в вызов функции `localise` можно передать дополнительные параметры: признак необходимости использовать асинхронный режим и ссылку на функцию, которая должна быть вызвана после загрузки всех файлов ❻. Если в очереди еще остались файлы, функция вызывает себя еще раз, чтобы загрузить следующий файл ❼.

После определения процедуры обработки единственного пакета выполняются определение кода языка и региона ❽ и загрузка всех пакетов по очереди ❾.

Чтобы провести в жизнь то, ради чего создавалось расширение, определяется вторая функция с локализованным именем оригинала[1]:

```
$.localize = $.localise;
```

Полный код расширения можно загрузить на веб-сайте книги.

6.2. Расширение Cookie

Еще одним отличным примером расширения-функции является расширение Cookie (https://github.com/carhartl/jquery-cookie), написанное Клаусом Хартлом (Klaus Hartl). Оно позволяет читать или

[1] Версия имени `localise` более характерна для британского английского, а версия `localize` (с буквой `z` вместо `s`) используется в американском английском. – *Прим. перев.*

писать данные cookies, связанные с веб-страницей, даже не зная ничего о формате и кодировке, используемых в этих cookies. С помощью cookies можно сохранять на компьютерах конечных пользователей некоторую информацию о состоянии веб-приложения для большего удобства. Это расширение также не связано с обработкой коллекций элементов.

6.2.1. Операции с данными cookie

Cookies – это небольшие блоки данных, хранящиеся на компьютерах пользователей и связанные с одной или несколькими веб-страницами. Информация в cookies доступна в связанной с ним веб-странице и передается серверу с каждым запросом, что позволяет поддерживать на клиентской машине некоторое подобие состояния веб-приложения. Cookies хранятся определенное время и затем автоматически удаляются браузером.

Чтобы установить cookie для текущей веб-страницы с помощью расширения Cookie, необходимо указать имя и значение. Например, чтобы проследить, было ли показано пользователю введение при первом посещении вашего сайта, можно сохранить cookie с этой информацией:

```
$.cookie('introShown', true);
```

Для настройки cookie можно также передавать дополнительные параметры: срок хранения (по умолчанию срок хранения истекает по окончании сеанса), домен и путь, к которому применяется, признак необходимости передачи в зашифрованном виде и признак кодирования значения:

```
$.cookie('introShown', true, {expires: 30, domain: 'example.com',
         path: '/', secure: true, raw: true});
```

Чтобы получить значение cookie, достаточно просто указать имя. Если cookie с таким именем не существует, функция вернет значение `null`. В примере с отображением вводной информации необходимо предварительно проверить это значение и отображать введение, только если функция вернет значение `null`:

```
var introShown = $.cookie('introShown');
```

Чтобы удалить cookie, следует присвоить требуемому имени значение `null`:

```
$.cookie('introShown', null);
```

Как видите, расширение следует принципу *использования единственного имени* (в пределах jQuery). В зависимости от числа аргументов и их типов расширением предоставляются разные функциональные возможности.

Теперь, когда вы познакомились с возможностями расширения Cookie, можно перейти к обсуждению его реализации. Несмотря на то что расширение Cookie не использует мою инфраструктуру поддержки расширений, оно все же имеет похожую организацию и следует некоторым принципам, обозначенным в главе 4.

6.2.2. Чтение и запись cookies

Подобно расширениям, рассматривавшимся выше, тело расширения Cookie заключено в функцию (листинг 6.5), защищающую его от внешнего мира JavaScript, и доступно только через объект `jQuery`.

Листинг 6.5. Инкапсуляция реализации расширения

```
// ❶ Объявление анонимной функции
(function($, document) {

    // ❷ Объявление главной функции расширения
    $.cookie = function(key, value, options) {
        ... // Здесь располагается остальной код
    };

})(jQuery, document); // ❸ Немедленный вызов функции
```

Анонимная функция ❶ скрывает реализацию расширения от внешнего мира JavaScript в соответствии с принципами *сокрытия тонкостей реализации с использованием областей видимости* и *отказа от предположения, что имя $ будет ссылаться на jQuery* ❸. Данное расширение практически не использует библиотеку jQuery и определяет единственную функцию – `cookie`, которая добавляется в объект `jQuery` ❷, в соответствии с еще двумя принципами: *создания единственного имени и использования только его для взаимодействий с расширением* и *размещения всего необходимого в объекте jQuery*.

Расширение имеет два режима работы – чтения и записи, – которые определяются количеством и типами аргументов. Сначала посмотрим, как реализована запись значения cookie (листинг 6.6).

Листинг 6.6. Запись значения cookie

```
// Если переданы аргументы key и value, записать значение cookie...
// ❶ Убедиться, что запрошена запись значения cookie
if (arguments.length > 1 && (!/Object/.test(
```

```
        Object.prototype.toString.call(value)) || value == null)) {
    options = $.extend({}, $.cookie.defaults, options);

    // ❷ Проверить необходимость удаления
    if (value == null) {
        options.expires = -1;
    }

    // ❸ Привести параметры к стандартному виду
    //    при необходимости присвоить значения по умолчанию
    if (typeof options.expires === 'number') {
        var days = options.expires,
            t = options.expires = new Date();
        t.setDate(t.getDate() + days);
    }

    value = String(value);

    // ❹ Записать значение cookie и выйти
    return (document.cookie = [
        encodeURIComponent(key), '=', options.raw ?
            value : encodeURIComponent(value),
        options.expires ? '; expires=' +
            options.expires.toUTCString() : '',
        // использовать атрибут expires,
        // так как max-age не поддерживается в IE
        options.path    ? '; path=' + options.path : '',
        options.domain  ? '; domain=' + options.domain : '',
        options.secure  ? '; secure' : ''
    ].join(''));
}
```

Если функции передается более одного параметра и второй параметр не является объектом, следовательно, пользователем запрошена операция записи cookie ❶. При передаче значения `null` выполняется операция удаления. В этом случае время истечения срока хранения ❷ устанавливается равным `-1`. Это означает, что срок хранения cookie уже истек и браузер должен удалить его. В противном случае, если в параметре `expires` передается число, оно интерпретируется как количество дней, начиная от текущей даты ❸. В заключение значение cookie преобразуется в строку и выполняется запись cookie в браузер ❹. Как и следовало ожидать, далее применяется принцип *использования осмысленных значений по умолчанию*, то есть в cookie устанавливается текущий домен и путь, а также срок хранения, истекающий по окончании сеанса. В заключение функция возвращает закодированные имя и значение cookie, хотя в большинстве случаев вам не придется производить какие-либо операции с возвращаемым значением.

Если была запрошена операция чтения cookie, расширение продолжает обработку и возвращает его значение, как показано в листинге 6.7.

Листинг 6.7. Чтение значения cookie

```
// Если передан аргумент key и, возможно, options, прочитать cookie...
// ❶ Обработать параметры в аргументе options
options = value || $.cookie.defaults || {};
var decode = options.raw ? raw : decoded;
// ❷ Отделить значения в cookie
var cookies = document.cookie.split('; ');
// ❸ Извлечь значение из cookie
for (var i = 0, parts;
         (parts = cookies[i] && cookies[i].split('='));
          i++) {
    if (decode(parts.shift()) === key) {
        return decode(parts.join('='));
    }
}
// ❹ Вернуть null, если имя не найдено
return null;
```

Если во втором аргументе передаются параметры настройки, следовательно, функция вызвана для чтения cookie ❶. Если какой-то из параметров настройки не был указан явно, устанавливается значение по умолчанию. Далее функция извлекает текущие значения cookie (все cookies для данной веб-страницы) и разбивает их на пары ключ/значение ❷. Затем проверяются все пары в поисках указанного имени ❸ и возвращается соответствующее значение. Если искомое имя не найдено, возвращается значение `null` ❹.

Установить значения по умолчанию для параметров во всех cookies можно, изменив `$.cookie.defaults`:

```
$.extend($.cookie.defaults, {expires: 7});
```

Полный код расширения Cookie доступен для загрузки на сайте книги.

Это нужно знать

Расширения-функции создаются с целью добавления новых возможностей, не связанных напрямую с обработкой выбранных элементов.
Подключение новой функциональности производится непосредственно к точке интеграции `$`.
Защищайте свои расширения и предотвращайте конфликты имен с помощью создания областей видимости.
Предоставляйте параметры настройки в своих расширениях, но всегда инициализируйте их осмысленными значениями по умолчанию.

Попробуйте сами

Напишите расширение-функцию для форматирования времени, чтобы ее можно было использовать, как показано ниже:

```
var time = $.formatTime(new Date(0, 0, 0, 12, 34, 0));
```

Она должна принимать объект `Date` и извлекать время из него. При вызове без параметров она должна форматировать значение текущего времени. Время должно представляться в формате ЧЧ:ММ. Чтобы усложнить пример, добавьте поддержку необязательного первого параметра (`Boolean`), который позволял бы переключаться между 12- (по умолчанию) и 24-часовым форматами.
Подсказка: объект `Date` имеет функции `getHours` и `getMinutes`. Текущие дату/время можно получить вызовом `new Date()`.

6.3. В заключение

В отличие от расширений коллекций, расширения-функции не предназначены для работы с коллекциями элементов. Они реализуют вспомогательные функции, используемые для выполнения разнообразных операций в веб-странице. Они скрывают за своим фасадом порой весьма запутанные операции и позволяют не волноваться о различиях между браузерами.

Два примера, представленные в этой главе, – расширения Localization и Cookie – демонстрируют, как на практике создаются расширения-функции, как при создании расширений нового типа можно использовать инфраструктуру поддержки расширений и почему следует придерживаться основных принципов проектирования расширений.

После создания расширения необходимо убедиться в его работоспособности и что другие легко смогут получить расширение и понять, как им пользоваться. В следующей главе мы исследуем приемы тестирования, упаковки, развертывания и описания расширений, которые применяются в процессе подготовки расширений к их использованию обширным сообществом jQuery.

Глава 7

Тестирование, упаковка и документирование расширений

Эта глава охватывает следующие темы:

- тестирование расширений;
- упаковка расширений и подготовка к распространению;
- документирование расширений и демонстрация их возможностей.

Написав новое расширение, вы наверняка пожелаете сделать его доступным для широкого круга пользователей jQuery. Чтобы увеличить шансы на победу в конкурентной борьбе с другими расширениями, предоставляющими аналогичные функциональные возможности, необходимо убедиться, что расширение работает правильно во всех возможных ситуациях. Применение инструментов тестирования, таких как QUnit, позволит вам создать коллекцию тестов для проверки работы расширения в различных ситуациях.

Кроме того, расширение необходимо упаковать и включить в пакет все, что необходимо для использования расширения. Вместе с программным кодом в пакет должны включаться таблицы стилей, изображения, файлы локализации и, возможно, простые демонстрационные страницы. Чтобы уменьшить требования к пропускной способности сети, неплохо включить в пакет минифицированную версию расширения, которую можно создать с помощью одного из множества свободно распространяемых инструментов. Все файлы,

входящие в пакет, желательно сжать архиватором в единственный файл для удобства распространения.

Наконец, вместе с расширением должна распространяться документация с его описанием, чтобы пользователи знали, чего ожидать от расширения и как адаптировать его под разные потребности. Описывайте в документации все параметры настройки, каждую функцию обратного вызова, которую можно зарегистрировать, и каждый метод, доступный для вызова. Необходимо также наглядно показать возможности расширения с помощью демонстрационной страницы, предпочтительно с соответствующими фрагментами кода, которые пользователи смогут скопировать и вставить в свои проекты.

В этой главе подробно рассказывается о каждой из этих проблем с целью помочь вам подготовить свое расширение к распространению и упростить его использование другими пользователями.

7.1. Тестирование расширений

Тестирование расширений перед выпуском выглядит вполне очевидным требованием, но конкретная реализация тестирования – целое искусство. С ростом числа параметров настройки, влияющих на поведение расширения, количество их комбинаций растет в геометрической прогрессии, и тестирование их всех быстро превращается в весьма трудоемкую задачу.

Для начала можно создать простую страницу и устанавливать параметры настройки вручную, чтобы проверить их влияние, но с ростом сложности расширения такой подход становится непрактичным и приводит к появлению ситуаций, не охваченных тестированием. Наличие комплекта модульных тестов позволяет преодолеть эту проблему, давая возможность выполнить всеобъемлющее тестирование, ничего не упустив из виду.

Кроме того, наличие стандартного комплекта тестов (созданного в соответствии с принципом, требующим *создавать комплекты повторимых тестов*) позволяет без опаски выполнять рефакторинг кода, так как с его помощью можно убедиться в работоспособности всех внесенных изменений.

В этом разделе рассказывается о том, что должно подвергаться тестированию и как писать тесты с применением инструмента QUnit. В качестве примера мы создадим модульные тесты для проверки расширения MaxLength, реализованного в главе 5.

7.1.1. Что тестировать?

В идеальном случае тестированию должно подвергаться все, что имеется в расширении, – все методы для всех комбинаций параметров настройки, применение расширения к разным элементам, находящимся в разных местах на странице и во всех основных браузерах (что также является одним из руководящих принципов разработки). Но для большинства расширений тестирование каждого параметра или метода в отдельности или даже небольших групп взаимосвязанных настроек – не самое практичное решение.

Для начала проверьте установку значений по умолчанию параметров для всех экземпляров расширения (`$.pluginname.setDefaults`), затем – установку и извлечение отдельных параметров и их групп (`$(selector).pluginname('option')...`). Далее проверьте создание (`$(selector).pluginname()`) и уничтожение (`$(selector).pluginname('destroy')`) экземпляра расширения, чтобы убедиться, что оно производит все необходимые модификации в дереве DOM и затем удаляет их полностью. Если расширение предусматривает возможность активации/деактивации (enabling/disabling), протестируйте ее (`$(selector).pluginname('disable')`). Убедитесь, что расширение не выполняет никаких операций, находясь в неактивном состоянии.

Проверьте действие каждого параметра в отдельности, чтобы убедиться в правильности их влияния на целевые элементы. Для параметров, которые могут принимать значения разных типов, добавьте тестирование каждого из этих типов. Для параметров, которые являются ссылками на функции-обработчики, добавьте тесты, проверяющие их вызов и передачу всех необходимых аргументов.

Проверьте работу всех методов и вспомогательных функций, предоставляемых расширением. Убедитесь, что методы, которые, как предполагается, допускают включение в цепочки вызовов, не возвращают недопустимые значения.

Проверьте реакцию элементов, находящихся под управлением расширения, с помощью методов генерации событий jQuery. Сгенерируйте обычное всплывающее событие (которое передается вмещающему элементу в процессе всплытия вверх по дереву DOM) вызовом `$(selector).trigger('eventname')`. Чтобы сгенерировать невсплывающее событие, используйте метод `$(selector).triggerHandler('eventname')`. Для возбуждения событий можно также использовать специализированные функции jQuery, такие как `$(selector).click()`. Если вместе с событием потребуется передавать дополни-

тельную информацию, такую как позицию указателя мыши в момент щелчка или код нажатой клавиши, используйте функцию `trigger`, но вместо имени события передавайте ей объект `Event`.

```
var e = $.Event('click');
e.pageX = 10;
e.pageY = 20;
$(selector).trigger(e);
```

7.1.2. Использование QUnit

QUnit – инструмент тестирования программного кода на JavaScript, разработанный Джоном Резигом (John Resig) и в настоящее время поддерживаемый командой jQuery (http://qunitjs.com/). Он используется разработчиками библиотек jQuery и jQuery UI для нужд тестирования самих этих библиотек и может применяться для тестирования любого программного кода на JavaScript. Вы можете создавать свои тесты, действующие под управлением QUnit, для проверки функциональности своего расширения.

Чтобы создать тест для QUnit, воспользуйтесь шаблоном разметки HTML, представленным в листинге 7.1. Эта разметка загружает реализацию и стили CSS инструмента QUnit, а также ваши сценарии, выполняющие тестирование, которые должны подключаться в виде отдельных элементов `script`. В теле страницы имеются два элемента `div`: один предназначен для вывода результатов тестирования (`#qunit`), а другой – для размещения тестируемых элементов (`#qunitfixture`). Последний скрыт из поля зрения перемещением за границы окна, что обеспечивает имитацию «видимости» содержимого без захламления окна браузера во время тестирования. На рис. 7.1 показаны результаты загрузки примера страницы из листинга 7.1 – тест завершился неудачей из-за несовпадения фактического значения с ожидаемым.

Листинг 7.1. Шаблон страницы для QUnit

```
<html>
<head>
  <title>QUnit basic example</title>
  <link rel="stylesheet"
    href="http://code.jquery.com/qunit/qunit-git.css"/>
  <script src="http://code.jquery.com/qunit/qunit-git.js"></script>
  <script>
    test('a basic test example', function() {
      var value = 'hi';
      equal(value, 'hello', 'We expect value to be hello');
    });
```

```
  </script>
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
</body>
</html>
```

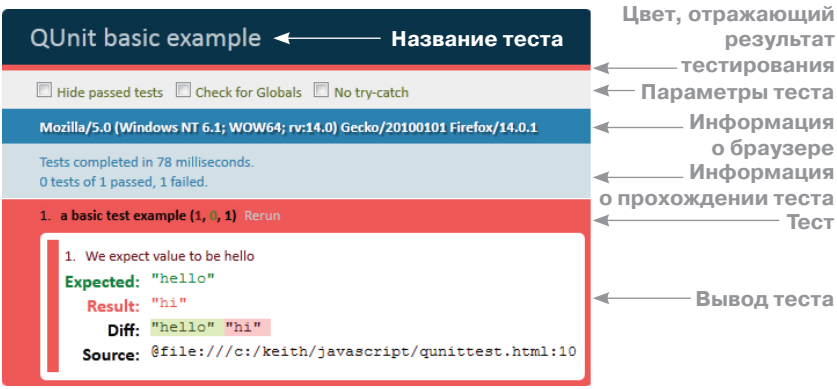


Рис. 7.1 ❖ Результаты выполнения простейшего примера использования QUnit

Реализация комплекта тестов состоит из одного или более вызовов функции `test`, каждый из которых содержит группу взаимосвязанных *утверждений* (assertions – инструкций, проверяющих фактические результаты). Каждый раздел включает код настройки окружения и одно или более утверждений, проверяющих результаты тестирования. Каждый раз, когда вызывается функция `test`, она заново воссоздает тестовое окружение на основе содержимого элемента `div` с идентификатором `#qunit-fixture`, защищая тем самым тесты от взаимовлияния и обеспечивая запуск каждого из них в заранее известном состоянии.

В заголовке страницы присутствует ее название, цветная полоса, отражающая результат тестирования (в случае успеха окрашивается в зеленый цвет, а в случае неудачи – в красный), идентификационная информация о браузере и несколько флажков, влияющих на поведение теста. Информация с результатами тестирования отображается в теле страницы. По умолчанию успешно выполнившиеся тесты сворачиваются, а потерпевшие неудачу разворачиваются и отображают информацию об отдельных утверждениях. Тест можно свернуть или развернуть щелчком на его заголовке. Щелчок на ссылке **Rerun (По-**

вторить) или двойной щелчок на заголовке теста вызовет повторное его выполнение.

Имеется также возможность фильтровать отдельные тесты, передавая фрагменты их названий (без учета регистра символов) в тестовую страницу в виде параметров, например:

```
test.html?filter=basic
```

Условие в фильтре можно инвертировать, добавив перед именем теста восклицательный знак (!):

```
test.html?filter=!event
```

Если отметить флажок **Check for Globals** (**Проверка наличия глобальных имен**) в заголовке, тест будет запущен повторно, но на этот раз при обнаружении хотя бы одной глобальной переменной будет генерироваться ошибка. Используйте этот параметр для мониторинга глобального пространства имен, чтобы избежать конфликтов с другими библиотеками. Если отметить флажок **No Try-Catch** (**Исключить инструкцию Try-Catch**), тест также будет запущен повторно, но на этот раз никакие исключения не будут перехватываться, что позволит им продолжить распространение вплоть до уровня браузера и остановить тестирование в этой точке. Обычно исключения перехватываются тестовым окружением и отображаются в виде сообщений об ошибках в элементе, предназначенном для вывода.

Попробуйте открыть свою тестовую страницу во всех основных браузерах, чтобы убедиться в совместимости со всеми этими платформами.

Чтобы показать на примере, как осуществляется тестирование расширений, мы создадим комплект тестов для расширения MaxLength из главы 5.

7.1.3. Тестирование расширения MaxLength

Чтобы убедиться в работоспособности расширения MaxLength, напишем комплект тестов QUnit it.

Сначала проверим функцию `setDefaults`. Убедившись, что начальное значение параметра настройки имеет ожидаемое значение, вызовем эту функцию, чтобы изменить параметр, и выполним проверку повторно. Сообщение, передаваемое в вызов функции проверки, выводится в случае выполнения утверждения, а в случае неудачи выводятся сравнивавшиеся значения. На рис. 7.2 показан результат благополучного выполнения нашего первого теста, а в листинге 7.2 приводятся разметка страницы и код реализации теста.

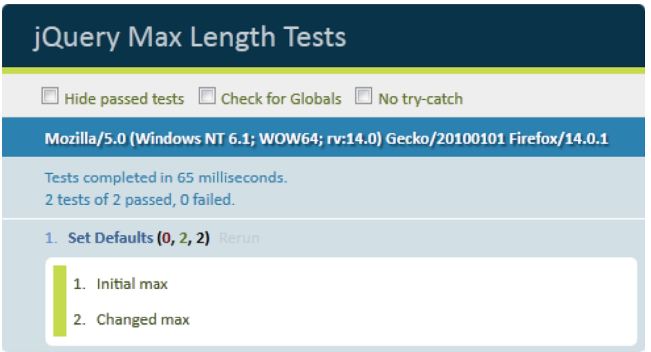


Рис. 7.2 ❖ Результат выполнения первого теста MaxLength

Листинг 7.2. Реализация тестов для расширения MaxLength

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<title>jQuery Max Length Tests</title>
<!-- ❶ Загрузка QUnit -->
<link type="text/css" rel="stylesheet"
      href="http://code.jquery.com/qunit/qunit-git.css">
<script type="text/javascript"
        src="http://code.jquery.com/qunit/qunit-git.js"></script>
<!-- ❷ Загрузка библиотеки jQuery и расширения -->
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js">
</script>
<script type="text/javascript" src="js/jquery.maxlength.js"></script>
<script type="text/javascript">
$(function() {
    // ❸ Определение теста
    test('Set Defaults', function() {
        // ❹ Количество утверждений
        expect(2);
        init();
        // ❺ Проверить утверждение
        equal($.maxlength._defaults.max, 200, 'Initial max');
        $.maxlength.setDefaults({max: 300});
        equal($.maxlength._defaults.max, 300, 'Changed max');
        $.maxlength.setDefaults({max: 200});
    });
});
function init(settings) {
```

```
    return $('#txa').val('').maxlength('destroy').maxlength(settings);
}
</script>
</head>

<body>
<div id="qunit"></div>
<div id="qunit-fixture">
    <input type="text" id="fbk1"><span id="fbk2"></span>
    <textarea id="txa" rows="3" cols="30"></textarea>
</div>
</body>
</html>
```

Тестовая страница начинается с загрузки стилей и кода QUnit ❶, а также библиотеки jQuery и расширения ❷.

Тело страницы содержит два стандартных элемента `div`: первый (`#qunit`) является интерфейсом QUnit, где отображаются результаты тестирования ❻, а второй (`#qunit-fixture`) хранит элементы, необходимые для тестирования ❼. Последний из них клонируется и воссоздается заново для каждого теста, чтобы обеспечить свежее тестовое окружение, и смещается за границы видимой области, чтобы его содержимое оставалось «видимым» и не перекрывало интерфейс QUnit с результатами.

Сами тесты определяются в вызовах функции `test` ❸. Функции передаются название комплекта утверждений и функция, выполняющая их. Каждый тест должен проверять определенный аспект расширения и может быть выполнен отдельно, для проверки отдельной особенности. Любые изменения в окружении, производимые в процессе тестирования, будут действовать до завершения текущего вызова функции `test`.

Чтобы убедиться, что все ошибки в коде корректно обработаны, каждый тест должен начинаться с вызова функции `expect`, определяющего число утверждений ❹. Тем самым, если возникнет какая-либо проблема, препятствующая выполнению всех утверждений, QUnit сможет проинформировать вас об ошибке, сравнив число фактически выполненных утверждений с заявленным. Число утверждений можно также передать во втором параметре функции `test`, переместив функцию обратного вызова в третью позицию. Наконец, необходимо инициализировать окружение, в котором будет выполняться тестирование, и определить сами утверждения ❺.

Чтобы протестировать функцию `setDefaults`, необходимо вызвать ее для изменения значения параметра и проверить, было ли произве-

дено изменение в действительности. Вызов функции `equal` является утверждением QUnit. Она сравнивает фактическое значение с ожидаемым (первые два аргумента) и сообщает об успехе, если значения равны (после преобразования типов, если потребуется). Сообщение, передаваемое в вызов `equal` в третьем аргументе, используется для регистрации результата.

Функция `init` производит инициализацию поля ввода `textarea` в элементе `div` с идентификатором `#qunit-fixture`, переустанавливая его значение, удаляя существующий экземпляр MaxLength и затем добавляя новый. В данном конкретном тесте, проверяющем глобальную функциональность, применение функции `init` необязательно, но она будет использоваться в реализациях следующих тестов. Элементы `input` и `span` внутри элемента `div` с идентификатором `#qunit-fixture` будут использоваться для вывода подсказки в последующих тестах.

7.1.4. Тестирование установки и извлечения параметров расширения

Создав тест для проверки установки значений параметров по умолчанию, можно перейти к тестированию функции установки и извлечения параметров настройки экземпляра. Установку отдельных параметров или их коллекций можно выполнить одним вызовом. Аналогично извлечь один или все параметры также можно одним вызовом. Все эти возможности необходимо протестировать, как показано в листинге 7.3.

Листинг 7.3. Тестирование установки и чтения параметров экземпляров MaxLength

```
// ❶ Определение теста параметров настройки расширения
test('Options', function() {
    // ❷ Число утверждений в тесте
    expect(12);
    // ❸ Инициализировать тестовый элемент
    var txa = init();
    // ❹ Проверить утверждение об объекте
    deepEqual(txa.maxlength('option'), {max: 200, truncate: true,
        showFeedback: true, feedbackTarget: null,
        feedbackText: '{r} characters remaining ({m} maximum)',
        overflowText: '{o} characters too many ({m} maximum)',
        onFull: null}, 'Initial settings');
    // ❺ Проверить утверждение о значении
    equal(txa.maxlength('option', 'max'), 200,
        'Initial max setting');
    equal(txa.maxlength('option', 'truncate'), true,
```

```
        'Initial truncate setting');
    // ❻ Изменить параметр и проверить еще раз
    txa.maxlength('option', {feedbackText: 'Used {c} of {m}'});
    deepEqual(txa.maxlength('option'), {max: 200, truncate: true,
        showFeedback: true, feedbackTarget: null,
        feedbackText: 'Used {c} of {m}',
        overflowText: '{o} characters too many ({m} maximum)',
        onFull: null}, 'Changed settings');
    equal(txa.maxlength('option', 'max'), 200,
        'Unchanged max setting');
    equal(txa.maxlength('option', 'truncate'), true,
        'Unchanged truncate setting');
    // ❼ Проверить изменение сразу нескольких параметров
    txa.maxlength('option', {max: 100, showFeedback: false});
    deepEqual(txa.maxlength('option'), {max: 100, truncate: true,
        showFeedback: false, feedbackTarget: null,
        feedbackText: 'Used {c} of {m}',
        overflowText: '{o} characters too many ({m} maximum)',
        onFull: null}, 'Changed settings');
    equal(txa.maxlength('option', 'max'), 100,
        'Changed max setting');
    equal(txa.maxlength('option', 'truncate'), true,
        'Unchanged truncate setting');
    // ❽ Проверить изменение отдельного параметра
    txa.maxlength('option', 'truncate', false);
    deepEqual(txa.maxlength('option'), {max: 100, truncate: false,
        showFeedback: false, feedbackTarget: null,
        feedbackText: 'Used {c} of {m}',
        overflowText: '{o} characters too many ({m} maximum)',
        onFull: null}, 'Changed named setting');
    equal(txa.maxlength('option', 'max'), 100,
        'Unchanged max setting');
    equal(txa.maxlength('option', 'truncate'), false,
        'Changed truncate setting');
});
```

Здесь определяется новый тест для проверки операций с параметрами настройки ❶. Как и в предыдущем случае, указывается число утверждений в тесте, чтобы гарантировать, что ни одно из них не будет пропущено по ошибке ❷.

Вызовом функции `init` производится инициализация тестируемых элементов ❸, после чего производится проверка соответствия начального состояния ожидаемому. Вызов метода `option` без параметров возвращает объект с текущими значениями параметров настройки. С помощью функции `deepEqual` из библиотеки QUnit выполняется сравнение этого объекта с ожидаемым ❹. Эта функция отличается от `equal` тем, что сравнивает все атрибуты двух объектов по отдель-

ности (и рекурсивно, если потребуется) вместо простого сравнения ссылок на эти объекты. Функция `equal` ❺ используется для проверки значения единственного параметра, извлеченного с помощью метода `option`.

Далее тест изменяет значение одного параметра вызовом того же метода `option` ❻ и проверяет результат изменения. Также проверяется ситуация одновременного изменения нескольких параметров, передачей коллекции новых значений методу ❼, и в заключение проверяется изменение отдельного параметра по его имени ❽.

7.1.5. Имитация действий пользователя

Поведение расширения MaxLength зависит от действий, выполняемых пользователем. В частности, от ввода текста в целевой элемент `textarea`. Поведение расширения можно проверить, сымитировав подобные действия. При тестировании других расширений может потребоваться сымитировать щелчки мышью или буксировку элементов мышью. В листинге 7.4 показано, как проверить реакцию MaxLength на действия пользователя.

Листинг 7.4. Тестирование ввода текста

```
// ❶ Определение теста, проверяющего реакцию на ввод текста
test('Text', function() {
    expect(28);
    var txa = init({max: 20});
    var rem = txa.nextAll('.maxlength-feedback');
    // ❷ Сымитировать ввод текста
    keyboard(txa, 'abcdefghij');
    // ❸ Проверить введенный текст
    equal(txa.val(), 'abcdefghij', 'Entered short text');
    // ❹ Проверить логическое утверждение
    ok(!txa.hasClass('maxlength-full'), 'Not full with short text');
    ok(!txa.hasClass('maxlength-overflow'),
        'Not overflow with short text');
    equal(rem.text(), '10 characters remaining (20 maximum)',
        'Feedback for short text');
        keyboard(txa, 'klmnopqrstuvwxyz');
    equal(txa.val(), 'abcdefghijklmnopqrst', 'Entered full text');
    ok(txa.hasClass('maxlength-full'), 'Full with full text');
    ok(txa.hasClass('maxlength-overflow'), 'Not overflow with full text');
    equal(rem.text(), '0 characters remaining (20 maximum)',
        'Feedback with full text');
    backspace(txa);
    equal(txa.val(), 'abcdefghijklmnopqrs', 'BS');
    ok(!txa.hasClass('maxlength-full'), 'Not full with BS');
    ok(!txa.hasClass('maxlength-overflow'), 'Not overflow with BS');
```

```
    equal(rem.text(), '1 characters remaining (20 maximum)',
        'Feedback with BS');
    keyboard(txa, 'u');
    equal(txa.val(), 'abcdefghijklmnopqrsu', 'More text');
    ok(txa.hasClass('maxlength-full'), 'Full with more text');
    ok(!txa.hasClass('maxlength-overflow'), 'Not overflow with more text');
    equal(rem.text(), '0 characters remaining (20 maximum)',
        'Feedback with more text');
    // Отключить усечение
    // ❺ Инициализировать с другими настройками и повторить тестирование
    txa = init({max: 20, truncate: false}).val('');
    ...
});

// ❻ Имитирует события от клавиатуры
function keyboard(input, chars) {
    for (var i = 0; i < chars.length; i++) {
        var ch = chars.charCodeAt(i);
        input.simulate('keydown', {charCode: ch}).
            simulate('keypress', {charCode: ch}).
            val(function(index, value) {
                return value + chars.charAt(i);
            }).
            simulate('keyup', {charCode: ch});
    }
}

// ❼ Имитирует клавишу "Забой" ("Backspace")
function backspace(input) {
    input.simulate('keydown', {keyCode: $.simulate.VK_BS}).
        simulate('keypress', {keyCode: $.simulate.VK_BS}).
        val(function(index, value) {
            return value.replace(/.$/, '');
        }).
        simulate('keyup', {keyCode: $.simulate.VK_BS});
}
```

Как и в примерах выше, здесь определяется тест, и ему присваивается определенное имя ❶. Далее следуют определение числа утверждений и инициализация расширения. Затем имитируется ввод текста с клавиатуры ❷ и проверяется содержимое поля ввода ❸. Инструмент QUnit позволяет проверять логические утверждения о состоянии расширения с помощью функции `ok` ❹. Эта функция принимает логическое значение в первом аргументе и сравнивает его со значением `true`. В данном случае проверяется отсутствие определенных классов в элементе `textarea`.

После выполнения последовательности операций и проверок производится повторная инициализация элемента `textarea` с другими

параметрами настройки и выполняются дополнительные операции и тесты с целью проверить изменение поведения расширения ❺.

Для имитации событий, обычно возбуждаемых пользователем, определяются две вспомогательные функции. Функция `keyboard` ❻ генерирует события `keydown`, `keypress` и `keyup` для каждого символа в указанной строке, а функция `backspace` ❼ делает то же самое, но для единственного символа забоя. Для отправки событий в обеих используется расширение Simulate (https://github.com/eduardolundgren/jquery-simulate).

Большинство событий, обычно возбуждаемых пользователем, легко можно сгенерировать с помощью стандартных функций из библиотеки jQuery. Например, событие щелчка мышью на элементе можно сымитировать вызовом функции `click`, `trigger` (которая генерирует всплывающее событие для всех элементов в коллекции) или `triggerHandler` (которая генерирует невсплывающее событие для первого элемента в коллекции).

```
$('#button1').click();
$('#button1').triggerHandler('click');
```

7.1.6. Тестирование функций-обработчиков

Многие расширения предусматривают возможность подключения пользовательских функций для обработки событий, таких как изменение значений или истечение времени ожидания. При наличии подобной возможности необходимо проверить условия возбуждения событий и параметры, передаваемые обработчикам. В расширении MaxLength поддерживается вызов обработчика при достижении или превышении длины текста в `textarea` установленного значения. В листинге 7.5 приводится реализация теста, проверяющего вызов обработчика событий.

Листинг 7.5. Тестирование обработчиков событий

```
// ❶ Инициализировать вспомогательные переменные
var count = 0;
var overflowing = null;
// ❷ Определение функции обратного вызова
function filled(overflow) {
    count++;
    overflowing = overflow;
}

// ❸ Определение теста, проверяющего вызов обработчика
test('Events', function() {
```

```
    expect(10);
    var txa = init({max: 20, onFull: filled});
    keyboard(txa, 'abcdefghijklmnopqrs');
    // ❹ Убедиться, что никаких событий не генерировалось
    equal(count, 0, 'No event');
    // ❺ Создать условие вызова обработчика
    keyboard(txa, 't');
    // ❻ Убедиться, что обработчик был вызван, и проверить его параметр
    equal(count, 1, 'Full event');
    equal(overflowing, false, 'Not overflowing');
    keyboard(txa, 'u');
    equal(count, 2, 'Full event');
    equal(overflowing, false, 'Not overflowing');
    // Отключить усечение
    // ❼ Инициализировать с другими настройками и повторить тестирование
    count = 0;
    overflowing = null;
    txa = init({max: 20, truncate: false, onFull: filled});
    keyboard(txa, 'abcdefghijklmnopqrs');
    equal(count, 0, 'No event');
    keyboard(txa, 't');
    equal(count, 1, 'Full event');
    equal(overflowing, false, 'Not overflowing');
    keyboard(txa, 'u');
    equal(count, 2, 'Full event');
    equal(overflowing, true, 'Overflowing');
});
```

Здесь сначала объявляется несколько переменных, помогающих отследить вызовы обработчика ❶, и определяется функция, которая фактически будет вызываться ❷. Для определения количества вызовов используются простой счетчик и еще одна переменная – для хранения единственного параметра.

Далее определяется сам тест ❸. Как и прежде, определяется число ожидаемых утверждений и инициализируются тестируемые элементы. Сначала проверяется несколько начальных утверждений, чтобы убедиться, что функция-обработчик не вызывается, когда она не должна вызываться ❹. Затем выполняется операция, создающая условие вызова обработчика ❺, и проверяется, действительно ли был вызван обработчик с требуемым значением параметра ❻.

Наконец выполняется повторная инициализация тестового окружения, изменяется параметр настройки и вновь выполняется проверка вызова обработчика в изменившихся условиях ❼.

Полную тестовую страницу для проверки расширения MaxLength можно загрузить на веб-сайте книги. Наряду с тестами, описанными выше, она включает тесты, проверяющие поведение расширения

в активном и неактивном состояниях, удаление расширения и вывод подсказки.

Итак, вы увидели, как с помощью пакета QUnit создавать модульные тесты, на примере расширения MaxLength. Однако QUnit обладает гораздо более широкими возможностями, чем было продемонстрировано в этой главе. Этот инструмент способен также группировать тесты в модули, производить тестирование в асинхронном режиме, предлагает дополнительные функции проверки утверждений и даже имеет свои точки подключения функций обратного вызова, с помощью которых можно следить за процессом тестирования. За дополнительной информацией о возможностях этого инструмента обращайтесь к документации с описанием QUnit API (http://api.qunitjs.com/), а также к статьям «Introduction» и «Cookbook» на главном сайте проекта (http://qunitjs.com/)[1].

7.2. Упаковка расширений

После проверки работоспособности расширения в разных ситуациях может возникнуть желание поделиться им с сообществом пользователей jQuery. Для этого необходимо упаковать все необходимое для нормальной работы расширения, чтобы упростить его распространение.

Проще говоря, необходимо собрать все файлы вместе, создать минифицированную версию программного кода с целью уменьшения времени его загрузки, реализовать для пользователей простой пример использования расширения и упаковать все файлы в один архив, чтобы упростить его передачу. Каждый из этих шагов описывается в данном разделе.

7.2.1. Сборка всех файлов вместе

Часто в состав расширения входит не только файл с программным кодом на языке JavaScript. Оно может также включать следующие файлы:

- дополнительные модули JavaScript с редко используемыми функциями;
- минифицированные версии модулей JavaScript (см. раздел 7.2.2);
- файлы локализации для поддержки других языков и стран;
- файлы CSS для оформления внешнего вида расширения;

---

[1] Из статей на русском языке можно порекомендовать http://ruseller.com/lessons.php?rub=32&id=761. – *Прим. перев.*

- изображения и другие ресурсы, которые могут использоваться посредством таблиц стилей или параметров настройки расширения;
- простой пример использования расширения (см. раздел 7.2.3);
- документация с описанием расширения (см. раздел 7.3).

Например, мое расширение Datepicker включает все эти файлы, кроме документации, которая доступна отдельно. Различные компоненты, входящие в состав моего расширения, можно видеть в списке файлов, представленном на рис. 7.3.

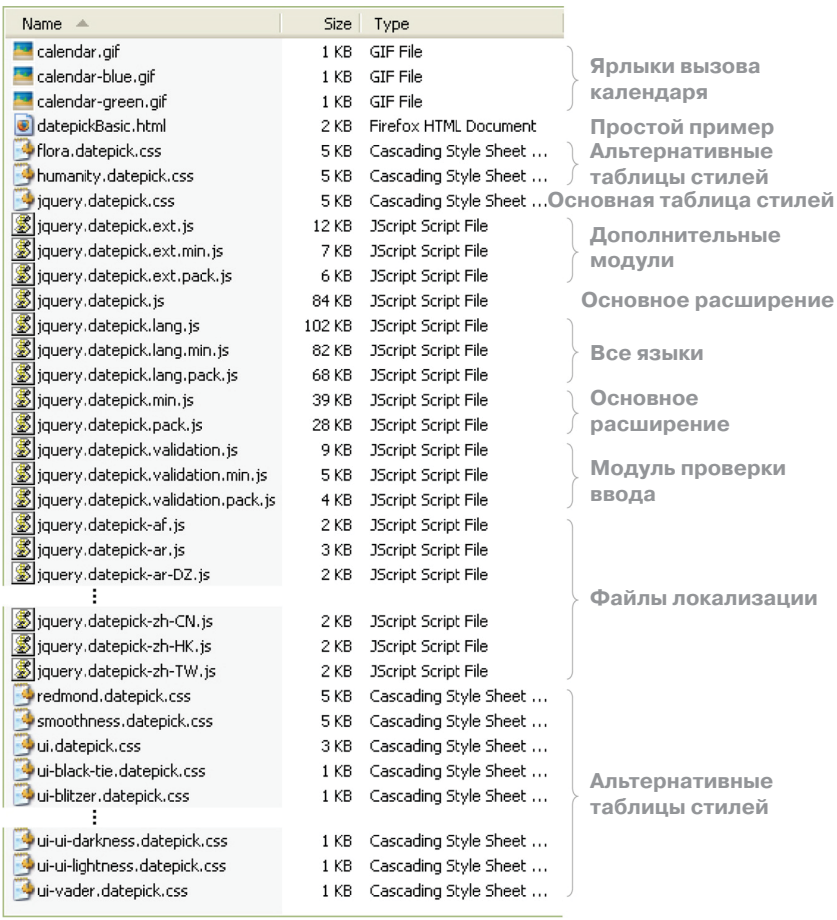


Рис. 7.3 ❖ Файлы, входящие в состав расширения Datepicker

Упакуйте свои файлы в zip-архив. Тем самым вы не только уменьшите размер пакета за счет сжатия, но и объедините все файлы в один пакет, упростите его распространение и избавитесь от беспокойства, что что-то забыли.

7.2.2. Минификация расширения

Чтобы уменьшить объем загрузки, можно сделать файл с программным кодом более компактным, удалив из него ненужный текст, такой как комментарии и пробельные символы. Этот процесс называется *минификацией* кода.

Включая в пакет обе версии, оригинальную и минифицированную, вы даете потенциальным пользователям выбор между версиями: для отладки и изучения можно использовать полную версию, а для промышленного использования – минифицированную. Файл с минифицированной версией должен иметь то же имя, что и файл с оригинальной версией, но с суффиксом `.min` после имени расширения, например `jquery.maxlength.min.js`.

После минификации кода в файл с минифицированной версией следует скопировать заголовочный комментарий из оригинальной версии, чтобы можно было установить название расширения, его версию и автора, и указать URL веб-сайта расширения, чтобы желающие смогли найти обновления, примеры и документацию.

В Сети можно найти специализированные минификаторы программного кода на JavaScript, например:

- минификатор Packer Дина Эдвардса (Dean Edwards) (http://dean.edwards.name/packer/);
- YUI Compressor (http://developer.yahoo.com/yui/compressor/);
- Google Closure Compiler (https://developers.google.com/closure/compiler/).

Packer

Минификатор Packer, написанный Дином Эдвардсом, – это веб-приложение, позволяющее сгенерировать стандартный минифицированный код, получаемый за счет удаления комментариев и пробельных символов. С его помощью можно также получить файл в кодировке Base62, размер которого обычно получается меньше, чем размер минифицированного кода, но требует дополнительной обработки на стороне клиента для восстановления оригинального кода. Кроме того, версия Base62 не сжимается архиваторами до такой же степени, как простая минифицированная версия, поэтому комбина-

ция минификации с фильтром gzip дает лучшую производительность. В обоих случаях вам предоставляется возможность сократить имена переменных. Вдобавок к дополнительному уменьшению размера файла эта возможность обеспечивает некоторую обфускацию кода.

Чтобы воспользоваться минификатором Дина Эдвардса, перейдите по указанному выше адресу, вставьте код в верхнюю область ввода, выберите желаемые настройки и щелкните на кнопке **Pack (Упаковать)**, как показано на рис. 7.4. Затем скопируйте результат из нижнего поля и сохраните в локальном файле.

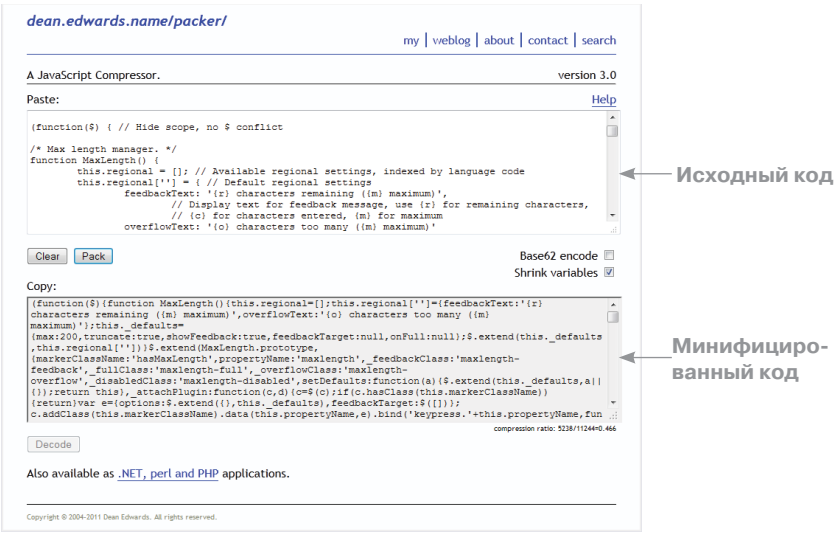


Рис. 7.4 ❖ Минификатор Дина Эдвардса в действии

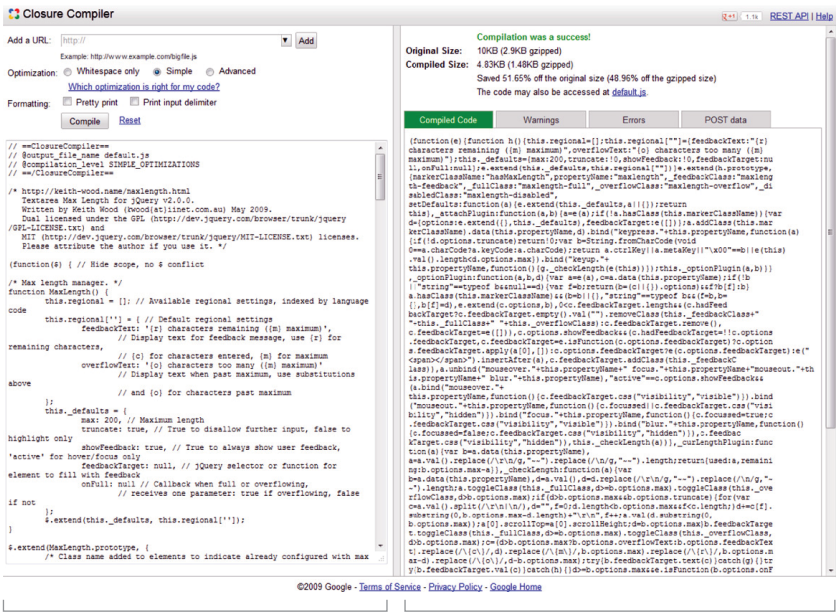
YUI Compressor

Минификатор YUI Compressor доступен в виде приложения на языке Java (1.4+), которое можно загрузить и запустить на локальном компьютере. Поддерживает возможность включения в локальный процесс сборки. Этот инструмент анализирует исходный код с помощью Rhino – реализации JavaScript на Java – и создает минифицированную версию, удаляя из нее комментарии и незначимые пробельные символы, одновременно замещая внутренние переменные более короткими значениями.

```
>java -jar \path\to\yuicompressor-2.4.7.jar -o jquery.maxlength.min.js
jquery.maxlength.js
```

Google Closure Compiler

Минификатор Google Closure Compiler использует иной подход. Помимо удаления ненужных комментариев и пробельных символов, он сканирует код на манер компилятора, с целью получить ту же функциональность меньшим объемом кода. Как побочный продукт он генерирует предупреждения и сообщения о (потенциальных) проблемах, найденных в коде. Компилятор доступен в виде веб-приложения (см. рис. 7.5) или в виде Java-приложения (http://closure-compiler.appspot.com/home).



Исходный код Минифицированный код

Рис. 7.5 ❖ Минификатор Google Closure Compiler в действии

Если вы решите воспользоваться веб-версией, вставьте исходный код в поле ввода слева, установите желаемые настройки и щелкните на кнопке **Compile** (Компилировать). Скопируйте результат из поля справа и сохраните в локальном файле.

Параметры настройки минификатора Google Closure Compiler

Минификатор Closure Compiler предлагает три уровня оптимизации. На самом простом уровне **Whitespace Only** (Только пробельные символы)

удаляются комментарии, символы перевода строки и ненужные пробельные символы. На следующем уровне **Simple** (**Простой**) удаляется все то же, что на предыдущем, плюс внутренним переменным присваиваются более короткие имена. Обе настройки без всякой опаски могут использоваться с любым кодом, если только вы не пытаетесь обращаться к переменным, используя имена в виде строковых значений.

На третьем уровне **Advanced** (**Расширенный**) выполняются оптимизации, описанные выше, а затем программный код анализируется с целью определить, возможно ли его переписать, чтобы достичь того же результата меньшим объемом кода. Для оптимизации на этом уровне необходимо, чтобы ваш программный код соответствовал определенным предположениям, которые делает компилятор, в противном случае его работоспособность может быть нарушена. За дополнительной информацией обращайтесь к разделу документации «Closure Compiler Compilation Levels» (https://developers.google.com/closure/compiler/docs/compilation_levels)[1].

Если включить параметр **Pretty Print** (**Форматированный вывод**), в результат будут возвращены символы перевода строки и отступы, чтобы сделать программный код удобочитаемым для человека, однако это увеличит размер файла. Если включить параметр **Print Input Delimiter** (**Выводить разделитель файлов**), в вывод будут добавлены комментарии, отмечающие начало каждого следующего файла.

Сравнение

Для сравнения результаты применения всех трех инструментов к расширению MaxLength приводятся в табл. 7.1. Как видите, различия между ними незначительные, особенно после сжатия архиватором zip. Но, как бы то ни было, Google Closure Compiler занял первое место.

Таблица 7.1. Результаты сравнения минификаторов

Минификатор	Размер файла после минификации (байт)	% экономии	После сжатия архиватором zip (байт)	% экономии
Packer	5238	53,4	1551	86,2
YUI Compressor	5192	53,8	1556	86,1
Google Closure Compiler	4949	56,0	1497	86,7

7.2.3. Реализация простого примера

Постарайтесь включить в состав пакета с расширением простой, но законченный пример его использования. Такой пример должен про-

[1] Описание уровней оптимизации на русском языке можно найти по адресам: http://javascript.ru/optimize/google-closure-compiler/simple-optimization и http://javascript.ru/optimize/google-closure-compiler/advanced-optimization. – *Прим. перев.*

демонстрировать пользователю возможности расширения, а также он поэкспериментирует с параметрами настройки и вызовами методов. Страница должна быть готова к работе сразу после распаковки пакета.

Сделайте страницу максимально простой, чтобы у пользователя не возникало сомнений в том, как использовать ваше расширение. Предусмотрите загрузку библиотеки jQuery (и jQuery UI, если необходимо) из сети доставки содержимого (CDN), чтобы не включать ее в свой пакет или беспокоиться о том, где она должна храниться относительно этой страницы. Задействуйте свое расширение с настройками по умолчанию и позвольте пользователям самим изменять параметры потом, когда они начнут изучать возможности расширения. Включите в эту страницу ссылку на главную демонстрационную страницу на вашем сайте и ссылку на документацию с описанием особенностей расширения.

Простая страница с примером использования расширения MaxLength представлена в листинге 7.6.

Листинг 7.6. Простая страница с примером использования расширения MaxLength

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<title>jQuery Max Length Basics</title>
<!-- ❶ Загрузить стили для расширения -->
<link type="text/css" href="jquery.maxlength.css" rel="stylesheet">
<!-- ❷ Загрузить библиотеку jQuery -->
<script type="text/javascript"
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js">
</script>
<!-- ❸ Загрузить расширение -->
<script type="text/javascript" src="jquery.maxlength.js"></script>
<script type="text/javascript">
$(function() {
    // ❹ Инициализировать расширение
    $('#maxLength').maxlength();
});
</script>
</head>
<body>
<h1>jQuery Max Length Basics</h1>
<p>This page demonstrates the very basics of the
    <!-- ❺ Ссылка не веб-сайт расширения -->
```

```
    <a href="http://keith-wood.name/maxlength.html">
    jQuery Max Length plugin</a>.
    It contains the minimum requirements for using the plugin and
    can be used as the basis for your own experimentation.</p>
<p>For more detail see the
    <a href="http://keith-wood.name/maxlengthRef.html">
    documentation reference</a> page.</p>
<p><span class="demoLabel">Default max length:</span>
    <!-- ❻ Целевой элемент -->
    <textarea id="maxLength" rows="5" cols="50"></textarea></p>
</body>
</html>
```

Сначала выполняется загрузка стилей расширения ❶, библиотеки jQuery ❷ и самого расширения ❸. Расширение вызывается с минимальным количеством настроек ❹. В помощь будущим пользователям включены ссылки на документацию и веб-сайт, где возможности расширения демонстрируются более полно ❺. Наконец, в странице присутствует элемент, к которому подключается расширение ❻, опять же, в самом простом виде.

Сеть доставки содержимого

Компания Google предоставляет собственную сеть доставки содержимого (Content Delivery Network, CDN) для распространения jQuery, откуда можно загрузить различные версии библиотеки, как полные исходные коды, так и минифицированные файлы. Измените номер версии на требуемый и уберите суффикс `.min`, если вам необходимы полные исходные коды:

```
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">
```

Из этой сети можно также загрузить библиотеку jQuery UI:

```
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.9.2/jquery-ui.
min.js">
```

и темы оформления ThemeRoller для jQuery UI. Измените номер версии и название темы на требуемые:

```
<link type="text/css"
  rel="stylesheet"
  href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.9.2/themes/
south-street/jquery-ui.css">
```

Компания Microsoft и команда разработки jQuery (посредством Media-Temple) тоже предоставляют свои сети CDN для распространения jQuery, jQuery UI и стандартных тем оформления ThemeRoller.

7.3. Документирование расширений

Расширение может обладать фантастическими возможностями и потрясающим богатством настроек, но если пользователи ничего не будут знать о его возможностях, они не смогут использовать все его богатства в полной мере. Большинство разработчиков добавляют комментарии в свой код (по крайней мере, должны делать это), а инфраструктура поддержки расширений собирает все параметры настройки воедино, но пользователи обычно не желают копаться в исходном коде в поисках подобных описаний.

Документируя расширения и выкладывая описание в Сети, вы помогаете пользователям оценить достоинства ваших расширений и адаптировать их под свои потребности. Ясная документация с примерами может существенно уменьшить бремя сопровождения, так как пользователи смогут самостоятельно найти ответы на интересующие их вопросы.

7.3.1. Документирование параметров настройки

Все параметры настройки, поддерживаемые расширением, должны быть описаны в документации. Для каждой настройки должны быть указаны имя параметра, его тип или типы, значение по умолчанию, а также описание его назначения и производимый им эффект. На рис. 7.6 изображена страница из документации к расширению MaxLength с описанием некоторых параметров.

Для наиболее сложных параметров, особенно принимающих объекты или функции, необходимо приводить фрагменты кода, иллюстрирующие их использование. Наряду с ожидаемым типом, значением по умолчанию и описанием перечислите также все атрибуты значения-объекта. Аналогично, если через параметр настройки передается функция, перечислите все ее аргументы, их допустимые значения, типы и назначение. Для функций следует также указать, на что ссылается переменная `this` в теле функции и что она должна возвращать.

Описывая взаимозависимости между связанными параметрами, добавляйте ссылки на них. Если число параметров достаточно велико, нелишним будет указать ссылки на них (в алфавитном порядке) в верхней и/или нижней части страницы, чтобы пользователь мог быстро переходить к описанию нужного параметра.

По мере дальнейшего развития расширения записывайте, в какой версии какой параметр был добавлен или изменен. Это поможет

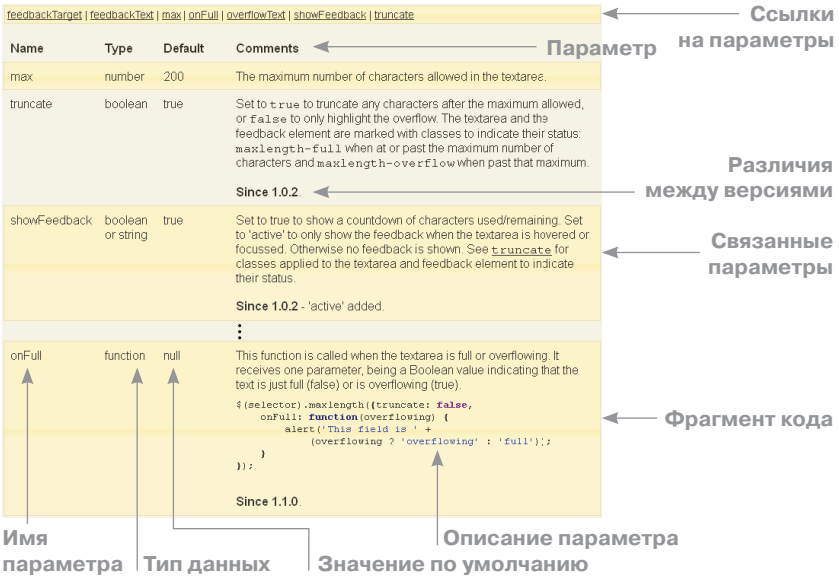


Рис. 7.6 ❖ Страница из документации к расширению MaxLength с описанием некоторых параметров

пользователям старых версий узнать, что следует сделать при переходе на более новую версию.

7.3.2. Документирование методов и вспомогательных функций

Документировать необходимо также все методы и вспомогательные функции, поддерживаемые расширением.

Для каждого метода или функции должны подробно описываться условия вызова, аргументы, возвращаемые значения, а также назначение функций и методов. Особо отмечайте все методы, не возвращающие объект jQuery, так как после них невозможно продолжение цепочки вызовов. На рис. 7.7 изображена страница из документации к расширению MaxLength с описанием нескольких первых функций и методов.

Перечислите все аргументы, укажите их типы, обязательность, значения по умолчанию и назначение. Предоставьте примеры вызова для каждой функции или метода. Включите в описание ссылки на другие методы или параметры настройки, если необходимо, и отметьте, в какой версии появился или изменился каждый метод или функция.

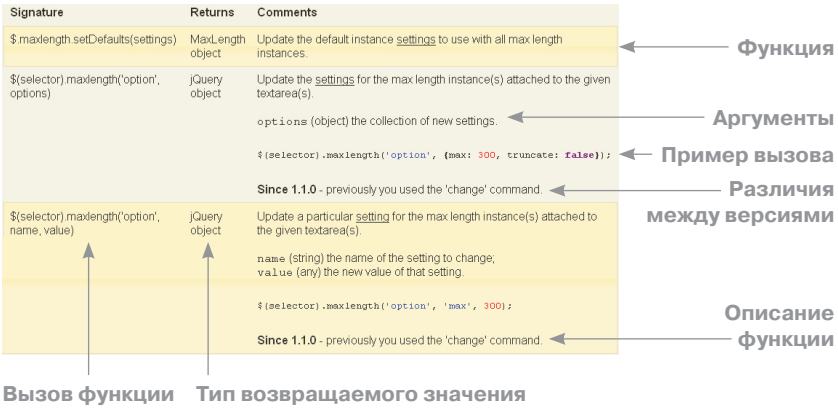


Рис. 7.7 ❖ Страница из документации к расширению MaxLength с описанием нескольких первых функций и методов

7.3.3. Демонстрация возможностей расширения

Первое впечатление играет очень важную роль, поэтому побеспокойтесь о том, чтобы представить свое расширение в выгодном свете, создав веб-страницу для демонстрации большинства или даже всех его возможностей.

В первую очередь продемонстрируйте работу расширения с настройками по умолчанию, а затем добавьте примеры, позволяющие изменять различные параметры. Если это возможно, включайте в демонстрацию фрагменты кода, соответствующие демонстрационным примерам, чтобы дать пользователям возможность отыскать пример, наиболее полно отвечающий их требованиям, и скопировать соответствующий код в свои проекты.

Демонстрационная страница может служить не только витриной и репозиторием примеров для потенциальных пользователей, но и ценным инструментом тестирования расширения, особенно его визуальных аспектов, которые сложно протестировать с применением инструментов автоматизации. Чтобы проверить совместимость расширения со всеми основными браузерами, достаточно будет просто открыть страницу в них и исследовать ее работоспособность.

На странице можно также разместить:

- инструкции для пользователей по включению расширения в страницу;
- отзывы других пользователей;

- список веб-сайтов, уже использующих расширение;
- краткий справочник по всем параметрам настройки;
- ссылку на документацию с более подробным описанием расширения;
- параметры доступа к файлам локализации (если необходимы);
- список предыдущих версий и изменений.

На рис. 7.8 изображен основной раздел демонстрационной страницы для расширения MaxLength.

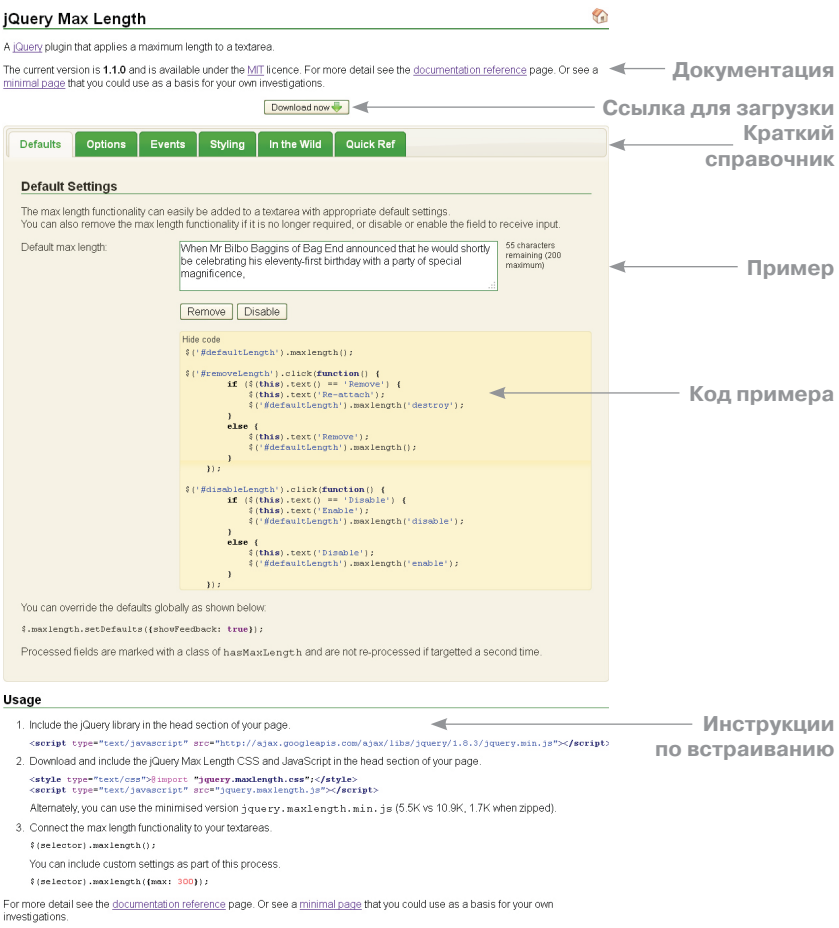


Рис. 7.8 ❖ Основной раздел демонстрационной страницы для расширения MaxLength

Это нужно знать

Наличие комплекта модульных тестов поможет гарантировать корректную работу расширения в большинстве ситуаций и в основных браузерах. Пакет QUnit – это фреймворк тестирования программного кода на JavaScript, позволяющий проверить все аспекты работы вашего расширения.
Упаковывайте все файлы, относящиеся к расширению, в один архив, чтобы упростить его распространение.
Включайте в дистрибутив минифицированную версию расширения, чтобы уменьшить сетевой трафик у потенциальных пользователей.
Документируйте все параметры настройки и методы своего расширения, чтобы помочь пользователям адаптировать его под свои нужды.
Создавайте демонстрационные страницы для своих расширений и добавляйте в эти страницы фрагменты с примерами кода, чтобы помочь пользователям опробовать его в своих условиях.

Попробуйте сами

Напишите тесты для расширения Watermark, созданного в главе 5, по образу и подобию тестов, созданных в этой главе для расширения MaxLength. Обязательно проверьте изменение параметра настройки при вызове метода `clear` для очистки метки.

7.4. В заключение

Вы можете написать самое лучшее расширение, но если пользователи не смогут с легкостью найти его, развернуть и настроить, они обратят свои взоры на другие предложения.

Пользователи ожидают, что в расширении не будет ошибок (по крайней мере, грубых) и оно будет широко доступно. Чтобы проверить работоспособность своего расширения, можно воспользоваться пакетом QUnit, позволяющим писать повторимые тесты. Постарайтесь проверить каждый параметр настройки, каждый метод и каждую функцию. Наличие тестов упрощает рефакторинг кода, давая возможность проверить правильную работу расширения сразу после внесения изменений.

Упаковывайте все файлы, относящиеся к расширению, в один zip-архив, чтобы упростить его распространение и гарантировать, что ни один из файлов не будет пропущен по забывчивости. Включайте в дистрибутив минифицированную версию кода, чтобы помочь пользователям снизить требования к пропускной способности сети.

Демонстрационная страница может служить полигоном для тестирования визуальных аспектов расширения и представить его в самом выгодном свете. Включайте в страницу фрагменты кода, чтобы дать пользователям возможность воспроизвести аналогичные эффекты в своих страницах.

Тщательно описывайте все особенности расширения в документации, чтобы пользователи могли познакомиться со всеми его возможностями и узнать, как настроить расширение под свои потребности. Описывайте все имеющиеся параметры настройки, включая обработчики событий, а также все методы и функции, поддерживаемые расширением. Исчерпывающая документация поможет вам облегчить бремя сопровождения, благодаря тому что пользователи смогут сами отыскать ответы на интересующие их вопросы.

В следующей части книги вы познакомитесь с инфраструктурой поддержки расширений в библиотеке jQuery UI и тем, как ее использовать для создания собственных расширений. Эта глава, описывающая тестирование, упаковку и документирование расширений, в равной степени относится к любым расширениям, создаваемым вами.

Часть III

Расширение jQuery UI

jQuery UI – это коллекция расширений пользовательского интерфейса, построенных на основе jQuery. Эти расширения реализуют функции, визуальные эффекты и виджеты для расширения возможностей веб-страниц. Библиотека jQuery UI имеет собственные точки интеграции и собственную инфраструктуру поддержки расширений.

В главе 8 мы займемся исследованием инфраструктуры виджетов jQuery UI. Эта инфраструктура также следует принципам разработки, описанным в предыдущей части, и помогает поддерживать единство внешнего оформления и внутренних возможностей во всех виджетах jQuery UI. В этой главе вы увидите пример создания законченного расширения на основе данной инфраструктуры.

Типичным требованием для виджетов пользовательского интерфейса является поддержка взаимодействий с мышью. Функция jQuery UI обеспечивает возможность такого взаимодействия посредством модуля Mouse. В главе 9 описывается, как интегрировать его функциональность в свои расширения.

Библиотека jQuery UI предоставляет также целую коллекцию визуальных эффектов для сокрытия или отображения элементов на странице с эффектами или для привлечения внимания к ним. В главе 10 вы узнаете, как создавать собственные эффекты, пользуясь возможностями jQuery UI, а также новые функции управления переходами (easing), то есть скоростью и ускорением изменения значений анимируемых атрибутов.

Глава 8

Виджеты jQuery UI

Эта глава охватывает следующие темы:

- что такое «виджеты jQuery UI»;
- использование инфраструктуры поддержки виджетов jQuery UI;
- следование принципам проектирования;
- создание законченного расширения для jQuery UI.

В предыдущей части книги вы видели, как создавать расширения коллекций с помощью инфраструктуры управления взаимодействиями с библиотекой jQuery. В этой части вы узнаете, что библиотека jQuery UI служит подобной основой, гарантирующей единообразный интерфейс доступа к внутренним возможностям библиотеки для всей коллекции расширений.

Библиотека jQuery UI – «это проверенный набор функций, виджетов, эффектов и тем оформления, работа которых основана на применении библиотеки jQuery» (http://jqueryui.com). Она является официальным дополнением к библиотеке jQuery, включающим несколько компонентов пользовательского интерфейса, известных как *виджеты*. Для обеспечения единообразия внешнего оформления всех виджетов, поддерживаемых библиотекой, в ней используется инструмент ThemeRoller (http://jqueryui.com/themeroller/).

Чтобы создавать визуальные компоненты, интегрирующиеся с существующими виджетами jQuery UI, ваши расширения должны опираться на эту инфраструктуру виджетов. Обе инфраструктуры, jQuery UI и инфраструктура, представленная в главе 5, позволяют сосредоточиться на создании уникальной функциональности, не отвлекаясь на низкоуровневые операции. Каждая из них позволяет хранить информацию о состоянии в элементах страницы и управлять параметрами настройки каждого экземпляра расширения, определяющими их внешний вид и поведение. Обе дают возможность выполнять дополнительные операции посредством приема вызова методов и передавать им все необходимые аргументы. И каждая из них может удалять экземпляры расширений, возвращая элементы страницы в исходное состояние.

Для сравнения с предыдущей инфраструктурой мы создадим еще одну версию расширения MaxLength, но уже в виде виджета jQuery UI. Напомню, что это расширение ограничивает длину текста, который можно ввести в поле ввода `textarea`, и предоставляет возможность вывода подсказки. К концу этой главы у вас в руках будет новое расширение, выглядящее как стандартный компонент jQuery.

8.1. Инфраструктура поддержки виджетов

Библиотека jQuery UI имеет модульную архитектуру, что позволяет выбирать, какие части пакета будут использоваться, и тем самым уменьшить объем загружаемого кода. Имеется даже возможность скомпоновать собственную версию библиотеки для загрузки с учетом зависимостей между модулями (http://jqueryui.com/download).

После обзорного знакомства с модулями, составляющими библиотеку jQuery UI, мы сосредоточимся на модуле Widget и приступим к созданию новой версии расширения MaxLength, опирающейся на инфраструктуру поддержки виджетов, которая входит в состав библиотеки.

8.1.1. Модули jQuery UI

Краткий обзор модулей, входящих в состав jQuery UI, поможет вам сориентироваться и познакомиться с доступными возможностями. Основная функциональность, используемая всеми виджетами jQuery UI, сосредоточена в модуле Core. Этот модуль всегда должен подключаться к странице. В нем имеется функция `zIndex`, с помощью которой можно узнать или изменить z-индекс элемента, чтобы расположить его поверх других элементов или спрятать за ними.

Модуль Widget – тема этой главы – предоставляет инфраструктуру поддержки, используемую всеми компонентами jQuery UI. Если виджет опирается в своей работе на взаимодействия с мышью, реализовать эти взаимодействия будет намного проще с помощью модуля Mouse, который позволяет определять свои обработчики для самых разных событий от мыши. Более подробно модуль Mouse исследуется в главе 9. Модуль Position – это самостоятельная утилита, упрощающая размещение элементов относительно друг друга.

Библиотека jQuery UI включает также несколько низкоуровневых модулей, реализующих взаимодействие с пользователем. Модуль Draggable позволяет организовать буксировку элементов страницы мышью, а модуль Droppable – определять элементы, способные принимать и обрабатывать буксируемые элементы после того, как они будут отпущены.

Для реализации изменения размеров элементов мышью можно использовать модуль Resizable. Модуль Selectable дает возможность выбирать элементы из списка по одному или группами. Чтобы получить возможность переупорядочивать элементы в списке, воспользуйтесь модулем Sortable, с помощью которого можно организовать перестановку элементов, буксируя их мышью из одной позиции в другую. В настоящее время в составе jQuery UI уже имеется несколько готовых виджетов (см. рис. 8.1), и в будущих версиях библиотеки планируется включить еще несколько.

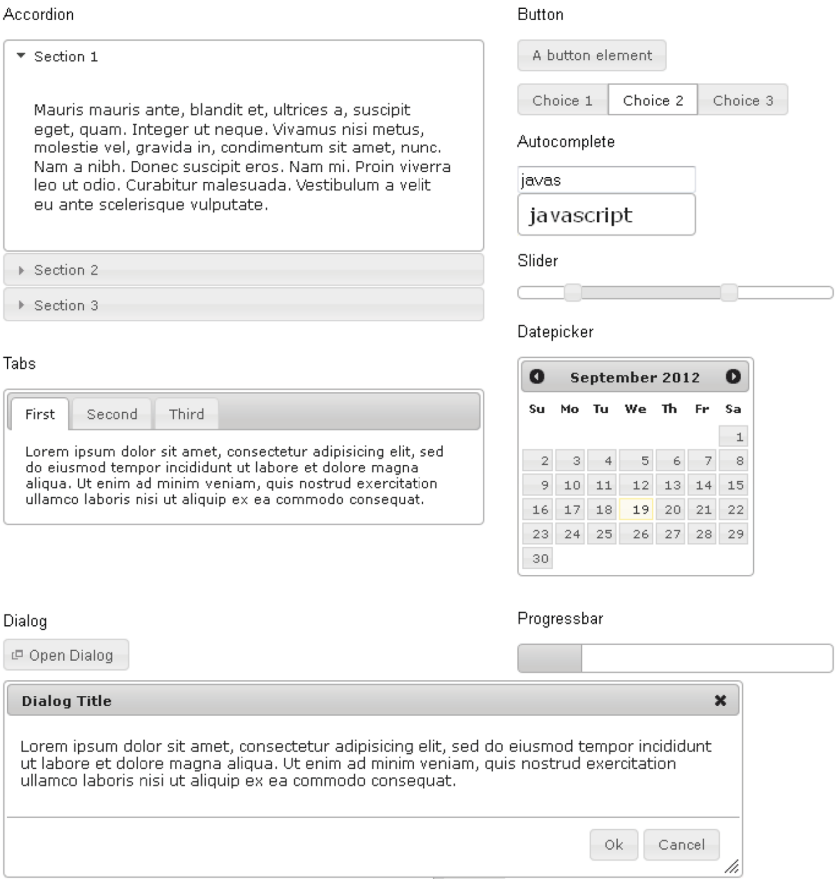


Рис. 8.1 ❖ Виджеты jQuery UI с поддержкой оформления посредством ThemeRoller

Модуль Accordion поможет разместить содержимое в панелях, сворачиваемых в заголовки по вертикали, а модуль Tabs – создать многостраничный интерфейс с вкладками, расположенными вдоль верхнего или нижнего края элемента. Для вывода диалогов можно использовать модуль Dialog и на время их отображения (при необходимости) делать все остальные элементы страницы недоступными.

Обеспечить единообразие внешнего вида и функциональности таких компонентов, как кнопки и ссылки, можно с помощью модуля Button. Модуль Autocomplete можно использовать для вывода возможных вариантов содержимого для поля ввода. Для организации ввода числовых значений или диапазонов в виде граничных значений можно использовать модуль Slider. Для выбора даты из всплывающего календаря можно использовать модуль Datepicker, можно также поместить этот календарь непосредственно в страницу, как статический элемент. Модуль Progressbar позволяет отображать ход выполнения некоторой операции.

В версии jQuery UI 1.9 появились: модуль Menu (реализующий средства навигации по сайту), модуль Spinner (позволяющий выбирать числовые значения кнопками «вверх» и «вниз») и модуль Tooltip (добавляющий всплывающие подсказки с настраиваемым внешним видом).

В составе jQuery UI имеется множество визуальных эффектов, дополняющих коллекцию простых эффектов из библиотеки jQuery. Многие из них, такие как Clip или Explode, могут применяться для создания интересных эффектов, сопровождающих появление или исчезновение элементов, другие, такие как Highlight и Shake, служат для привлечения внимания к элементам. Общая функциональность, совместно используемая многими эффектами, сосредоточена в модуле Effects Core. На рис. 8.2 показано действие нескольких эффектов на полпути к завершению.

Эффекты jQuery UI будут рассматриваться во всех подробностях в главе 10, а в этой главе мы займемся исследованием инфраструктуры jQuery UI поддержки виджетов, сосредоточенной в модуле Widget.

8.1.2. Модуль Widget

Модуль Widget является ключевым в разработке компонентов пользовательского интерфейса, предназначенных для использования совместно с библиотекой jQuery UI. Он определяет «класс» `$.Widget`,

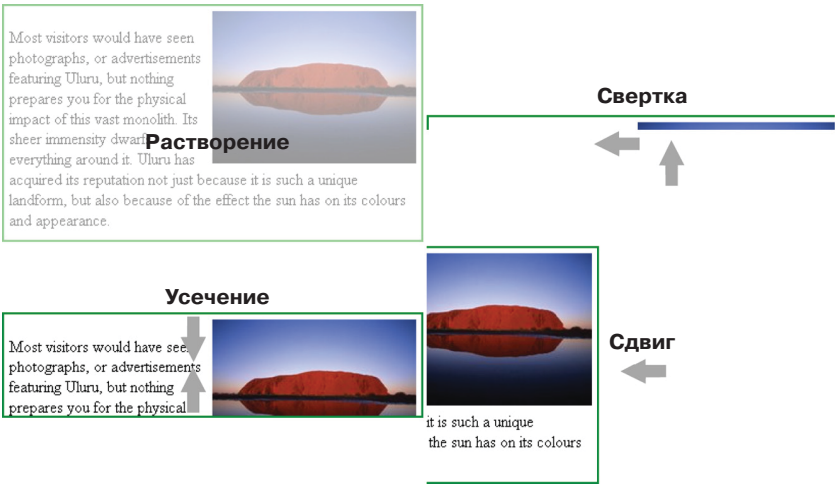


Рис. 8.2 ❖ Эффекты jQuery UI, выполненные на 60%. Слева сверху по часовой стрелке: растворение (fade), свертка (fold), сдвиг (slide) и усечение (clip)

являющийся основой любого нового компонента. (В JavaScript отсутствуют классы как формальные конструкции, однако есть возможность создавать объекты, напоминающие классы в других языках, и я буду называть их классами для простоты.) Этот класс реализует основные функциональные возможности, общие для всех виджетов, с целью обеспечить единообразие взаимодействий с ними: *инфраструктуру поддержки виджетов*.

В число стандартных возможностей инфраструктуры входят:

- подключение виджета к элементу;
- обработка параметров инициализации экземпляра виджета;
- извлечение и изменение значений настроек после инициализации;
- сохранение данных о состоянии виджета;
- обработка активного и неактивного состояний виджета;
- регистрация и вызов обработчиков событий;
- вызов собственных методов виджета;
- интеграция с инструментом ThemeRoller для придания единого внешнего вида;
- удаление виджета по окончании работы с ним.

Подробнее о жизненном цикле виджета (на примере расширения MaxLength) рассказывается в разделе 8.1.4.

8.1.3. Расширение MaxLength

Как упоминалось выше, в этой главе мы создадим новую версию расширения MaxLength, представленного в главе 5, но на этот раз с использованием инфраструктуры поддержки виджетов jQuery UI. Это расширение позволит накладывать ограничение на длину содержимого поля ввода `textarea`, подобно встроенному атрибуту `maxlength` в однострочных полях ввода. Вызываться оно будет, как и прежде, – с конкретными параметрами настройки либо с параметрами по умолчанию:

```
$('#text1').maxlength({max: 400});
$('#text1').maxlength();
```

Помимо ограничения длины текста, который можно ввести, расширение выводит подсказку с количеством введенных символов и оставшихся до достижения граничного значения, как показано на рис. 8.3. Для обеспечения единства внешнего вида с остальной страницей подсказка будет выводиться с применением темы ThemeRoller.

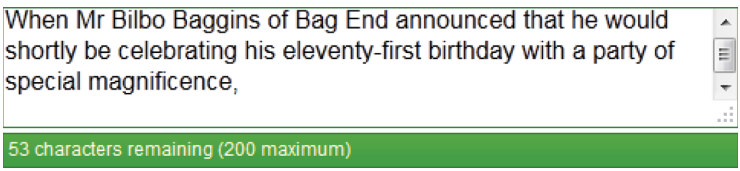


Рис. 8.3 ❖ Расширение MaxLength для jQuery UI в действии

Расширение будет включать те же функциональные возможности, что и прежде:

- вывод предупреждения при достижении максимально допустимого числа символов;
- отключение вывода подсказки;
- вывод подсказки, только когда поле ввода `textarea` владеет фокусом ввода;
- вызов пользовательской функции по достижении или превышении максимально допустимого числа символов.

Это расширение следует принципу *прогрессивного наращивания возможностей*. В браузерах с отключенной поддержкой JavaScript вы все еще сможете вводить текст, и ограничение длины будет осуществляться на стороне сервера. В браузерах с включенной поддержкой JavaScript расширение будет ограничивать длину текста перед отправкой формы и выводить подсказку.

8.1.4. Устройство расширения MaxLength

Расширение состоит из нескольких функций, переопределяющих или расширяющих возможности, предоставляемые инфраструктурой поддержки виджетов. Все они перечислены в листинге 8.1, а вслед за листингом описывается, когда вызывается каждая из них в течение жизненного цикла расширения.

Листинг 8.1. Функциональная структура расширения

```
var maxlengthOverrides = {
    // ❶ Настройки виджета по умолчанию
    options: {...},
    // ❷ Выполняет инициализацию элемента
    _create: function() {...},
    // ❸ Обрабатывает собственные параметры настройки
    _setOption: function(key, value) {...},
    // ❹ Обрабатывает собственные параметры настройки
    _setOptions: function(options) {...},
    // ❺ Перерисовывает виджет
    refresh: function() {...},
    // ❻ Возвращает текущую длину
    curLength: function() {...},
    // ❼ Применяет ограничение длины
    _checkLength: function() {...},
    // ❽ Удаляет расширение
    _destroy: function() {...}
};
$.widget('kbw.maxlength', maxlengthOverrides);
```

Как обычно, расширение применяется к одному или нескольким элементам `textarea` страницы, с использованием привычного для jQuery приема «выбирай и действуй». Инфраструктура поддержки виджетов создаст экземпляр объекта для каждого выбранного элемента и сохранит в нем информацию о текущем состоянии расширения. В этот экземпляр будут добавлены атрибуты по умолчанию (`options` в листинге 8.1 ❶) и дополнительные атрибуты, переданные в вызов функции инициализации. Эти атрибуты будут управлять внешним видом и поведением расширения. После выполнения всех необходимых настроек инфраструктура вызовет функцию `_create` ❷, в результате чего вы получите полностью готовый к работе экземпляр расширения.

Если позднее пользователь пожелает изменить настройки экземпляра расширения, он сможет воспользоваться методом `option`, который производит единственный вызов функции `_setOptions` ❹, если ему передается группа параметров настройки, и несколько вызовов функции `_setOption` ❸ при изменении отдельных параметров.

Обе процедуры, инициализации и установки параметров, завершаются вызовом функции `refresh` ❺, которая приводит внешний вид и поведение расширения в соответствие с новыми значениями параметров.

В ходе инициализации экземпляра расширения выполняется подключение обработчиков событий к элементу `textarea`, чтобы иметь возможность следить за нажатиями клавиш. При каждом нажатии любой клавиши выполняется вызов функции `_checkLength` ❻, которая применяет установленное ограничение на длину текста и обновляет элемент с подсказкой, отображая в нем обновленную информацию. Если длина текста в элементе `textarea` достигла предельного значения и пользователь зарегистрировал собственную функцию-обработчик данного события, он немедленно извещается об этой ситуации.

Пользователь в любой момент может получить текущую длину текста и количество символов, оставшихся до достижения предельного значения, вызвав метод `curLength`, который отображается в функцию `curLength` ❼:

```
var lengths = $('#text1').maxlength('curLength');
```

> Функции, имена которых не начинаются с символа подчеркивания (`_`) могут вызываться непосредственно, через инфраструктуру поддержки виджетов, для чего достаточно указать имя функции при обращении к расширению, как показано в примере выше. Функции, имена которых начинаются с символа подчеркивания, считаются скрытыми и недоступны непосредственно.

Пользователь может удалить экземпляр расширения MaxLength, вызвав метод `destroy`, который отображается в функцию `_destroy` ❽. В результате этого вызова будут отменены все изменения, произведенные на этапе инициализации.

Чтобы приступить к разработке расширения, его необходимо сначала объявить, дабы получить возможность пользоваться функциональностью инфраструктуры.

8.2. Определение виджета

Прежде чем приступить к разработке функциональных особенностей, характерных для конкретного виджета, следует выполнить несколько подготовительных операций:

- выбрать имя виджета:
- защитить код виджета от внешнего мира JavaScript, и внешний мир от виджета;
- объявить новый виджет.

Давайте рассмотрим их по порядку.

8.2.1. Выбор имени

Каждому расширению необходимо присвоить имя для его идентификации и отделения от других расширений. Инфраструктура поддержки виджетов помогает следовать принципам, требующим *ограничиваться единственным именем и использовать только его для взаимодействий с расширением*. Тем не менее вам все еще нужно выбрать имя для своего расширения и название пространства имен.

Пространство имен призвано помочь изолировать расширение от внешнего мира. Но, так как для доступа к функциональности расширения будет использоваться только одно имя, оно должно быть уникальным само по себе. Как уже говорилось раньше, если к странице будет подключено два расширения с одинаковыми именами, доступным окажется только одно из них (загруженное последним).

В данном примере мы будем использовать то же самое имя, что и прежде (`maxlength`), и добавим пространство имен `kbw` (это – мои инициалы). Имена расширений и пространств имен должны состоять только из символов нижнего регистра, цифр, дефисов (`-`) и подчеркиваний (`_`). Пространство имен `ui` официально зарезервировано для расширений jQuery UI и не должно использоваться для нужд сторонних расширений. Кроме того, название пространства должно указывать на разработчика расширения и подсказывать, является ли оно частью коллекции расширений.

Реализация расширения должна храниться в файле с именем `jquery.<имя_расширения>.js`. Чтобы как-то отличать данное расширение от расширения, созданного в главе 5, мы дадим файлу имя `jquery.maxlengthui.js`, оставив основное имя `maxlength` и добавив к нему окончание `ui`. Таблицы стилей CSS для расширения будут храниться в файле с тем же именем, но с другим расширением.

8.2.2. Инкапсуляция виджета

Хотя инфраструктура поддержки виджетов предоставляет множество функций, совместно используемых разными виджетами, нам все еще необходимо заключить реализацию своего виджета в анонимную функцию (листинг 8.2), как было показано в главе 5, чтобы соблюсти принципы *сокрытия тонкостей реализации с использованием областей видимости* и *отказа от предположения, что имя $ будет ссылаться на jQuery*. Это поможет защитить реализацию расширения от внешнего мира JavaScript, и наоборот.

Листинг 8.2. Инкапсуляция кода расширения

```
// ❶ Объявление анонимной функции
(function($) {
    ... здесь располагается остальной код
})(jQuery); // ❷ Немедленный вызов функции
```

Анонимная функция ❶ играет роль новой *области видимости* – переменные и функции, объявленные внутри, будут недоступны внешнему коду. Внутренний код не будет конфликтовать с внешним кодом. Для доступа к функциональности расширения вы сможете использовать объект `jQuery`, как обычно.

Функция-обертка гарантирует, что в ее теле идентификатор `$` будет ссылаться на объект `jQuery` за счет объявления параметра с именем `$` и последующей передачи `jQuery` в ее вызов ❷. Внутри этой функции находится код расширения, который начинается с объявления самого виджета.

> Как отмечалось выше, не следует обертывать код расширения вызовом `$(document).ready(function() {...})` или более короткой его формой `$(function(){...})`. Функция должна вызываться немедленно, сразу после загрузки, чтобы подготовить расширение к началу инициализации библиотеки jQuery.

8.2.3. Объявление виджета

Создание виджетов происходит под управлением инфраструктуры, что позволяет добавлять в них функциональные возможности, общие для всех виджетов. В число таких возможностей входят: инициализация и применение расширения к коллекции элементов, установка и извлечение параметров настройки, а также удаление экземпляров расширения, когда они становятся ненужными. В листинге 8.3 показано, как выполняется объявление виджета.

Листинг 8.3. Объявление виджета

```
// ❶ Определение функциональных возможностей виджета
var maxlengthOverrides = {

    // Глобальные параметры настройки расширения maxlength
    // ❷ Значения по умолчанию
    options: {
        max: 200, // Максимальная длина
        ... Другие значения параметров по умолчанию
    },

    _create: function() {...},
```

```
    // Остальные функции виджета
};

/* ❸ Объявление виджета Maxlength,
   накладывающего ограничения на элементы textareas.
   Зависит от jquery.ui.widget. */
$.widget('kbw.maxlength', maxlengthOverrides);
```

Инфраструктура поддержки виджетов дает возможность переопределять или расширять ее собственные функциональные возможности, поэтому сначала требуется определить объект с атрибутами, посредством которых будут переопределяться или расширяться эти функциональные возможности ❶, включая параметры настройки со значениями по умолчанию и собственные методы расширения. Содержимое этого объекта подробно будет описываться на протяжении оставшейся части данной главы.

Как и расширение из главы 5, данный виджет будет позволять настраивать его внешний вид и поведение посредством изменения значений параметров настройки. В соответствии с принципом *использования осмысленных значений по умолчанию* внутри объекта виджета необходимо определить все необходимые параметры настройки и присвоить им значения по умолчанию ❷. Большинство параметров данного виджета идентичны параметрам расширения MaxLength из главы 5. Единственное отличие – функция обратного вызова, которая должна вызываться при достижении или превышении установленного ограничения. В главе 5 атрибуту для подключения этой функции было дано имя `onFull`, но, согласно стандартной практике использования jQuery UI, в именах подобных атрибутов принято опускать префикс `on`, поэтому в данной реализации атрибут будет называться `full`. Отчасти такое решение обусловлено наличием в библиотеке jQuery UI механизма связывания обработчиков с такими внутренними событиями, реализованного в виде стандартных функций `bind` и `on`, который автоматически подключает обработчики к соответствующим событиям. Имя события формируется из имени расширения и имени параметра (атрибута). То есть подключить обработчик можно следующим образом:

```
$('#text1').maxlength().bind('maxlengthfull', function(event, ui) {
    ...
});
```

Библиотека jQuery UI не имеет стандартной стратегии локализации, поэтому мы не будем заниматься встраиванием поддержки локализации в данный виджет.

Виджет должен быть объявлен вызовом функции `$.widget` ❸, которой следует передать имя виджета и его пространство имен, разделенные точкой (.). В последнем аргументе передается объект, переопределяющий или расширяющий функциональные возможности, наследуемые виджетом.

Второй необязательный аргумент в этом вызове определяет, какие другие «классы» будут использоваться в качестве основы для данного виджета. Так как в данном случае создается новый виджет, наследующий только базовые функциональные возможности, этот аргумент был опущен, и по умолчанию виджет унаследует лишь «класс» `$.Widget`. В главе 9 описывается другой виджет для jQuery UI, использующий этот аргумент для подключения механизма взаимодействий с мышью путем передачи `$.ui.mouse` во втором аргументе.

В соответствии с объявлением виджета инфраструктура создаст новую функцию с именем, совпадающим с именем расширения (в данном случае `$.fn.maxlength`), и соединит ее с переопределениями, реализованными в вашем коде. За кулисами будет вызвана функция `$.widget.bridge`, которая свяжет ваш виджет с механизмом jQuery обработки коллекций.

Теперь, после объявления нового виджета, наследующего стандартную функциональность, можно приступать к реализации его поведения.

8.3. Применение к элементам

Инфраструктура поддержки виджетов автоматически применяет расширения к коллекциям элементов, но, возможно, у вас появится желание выполнить некоторые операции в тот момент, когда это будет происходить. Для этого необходимо переопределить унаследованную реализацию, которая по умолчанию ничего не делает.

8.3.1. Простое подключение и инициализация

Инфраструктура перехватывает вызов функции инициализации расширения и выполняет некоторые стандартные операции. Она создает экземпляр объекта и связывает его с выбранными элементами посредством функции `data`, добавляя атрибут с именем, соответствующим имени расширения. Атрибуту присваивается имя, составленное из названия пространства имен и названия расширения, разделенных дефисом – в версиях jQuery UI 1.9 и выше (в данном случае `kbw-maxlength`), или просто из названия расширения – в версиях, предшест-

вовавших jQuery UI 1.9 (maxlength). Внутри этого объекта создаются два дополнительных атрибута: element и options. Первый из них – это ссылка на элемент, к которому подключено расширение, а второй является копией параметров настройки расширения, значения которых определяются значениями по умолчанию и любыми параметрами настройки, переданными в вызов функции инициализации.

В процессе инициализации инфраструктура вызывает функцию _create, которую вы можете переопределить в своем расширении, как показано в листинге 8.4. Внутри функции _create ссылка this ссылается на текущий экземпляр объекта виджета.

Листинг 8.4. Подключение и инициализация

```
/* Инициализирует новый экземпляр виджета maxlength. */
// ❶ Переопределение функции _create
_create: function() {
    this.feedbackTarget = $([]);
    var self = this;
    // ❷ Добавить класс-метку
    this.element.addClass(this.widgetFullName || this.widgetBaseClass).
        // ❸ Добавить обработчики событий
        bind('keypress.' + this.widgetEventPrefix, function(event) {
            if (!self.options.truncate) {
                return true;
            }
            var ch = String.fromCharCode(event.charCode == undefined ?
                event.keyCode : event.charCode);
            return (event.ctrlKey || event.metaKey || ch == '\u0000' ||
                $(this).val().length < self.options.max);
        }).
        bind('keyup.' + this.widgetEventPrefix, function() {
            self._checkLength($(this));
        });
    // ❹ Перерисовать виджет
    this.refresh();
},
```

Чтобы инфраструктура поддержки виджетов могла вызвать ваш код в нужный момент времени, необходимо определить функцию _create ❶. Для идентификации целевых элементов к ним следует добавить класс-метку ❷. В главе 5 это делалось также с целью определить, было ли подключено расширение к тому или иному элементу. Инфраструктура jQuery UI, напротив, определяет факт подключения по присутствию экземпляра объекта виджета в данных элемента. Для идентификации она предоставляет атрибут this.widgetFullName (this.widgetBaseClass в версиях, предшествующих UI 1.9), значением кото-

рого является строка, состоящая из названия пространства имен и виджета, разделенных дефисом (-).

Остальной код в этой функции выполняет операции, характерные для данного расширения. Расширению MaxLength необходимо добавить в целевой элемент (`textarea`) обработчики событий от клавиатуры ❸, чтобы иметь возможность обновлять свое состояние синхронно с изменением текстового содержимого. Для упрощения идентификации событий позднее к их именам добавляется название пространства имен, которое можно получить из атрибута `this.widgetEventPrefix`, настраиваемого инфраструктурой. Обратите внимание, как обращением к атрибуту `options` виджета получить максимально допустимую длину для текущего элемента `textarea` и как вызвать внутреннюю функцию (`_checkLength`). И атрибут `options`, и функция `_checkLength` являются частями экземпляра объекта виджета `this`, который внутри обработчиков событий доступен через ссылку `self`.

В заключение необходимо вызвать собственную функцию `refresh` ❹, чтобы привести расширение в соответствие с текущими настройками. Подробнее об этой функции рассказывается в разделе 8.4.3, но прежде давайте посмотрим, как устанавливаются значения по умолчанию для параметров настройки и как инфраструктура реагирует на их изменение.

8.4. Параметры настройки

Изменить внешний вид или поведение экземпляра расширения можно передачей новых параметров настройки. Напомню, что *предоставление настроек* является одним из руководящих принципов. Попробуйте предугадать, что могут пожелать изменить пользователи, и предусмотреть соответствующие параметры. Затем позаботьтесь об удовлетворении принципа, касающегося *использования осмысленных значений по умолчанию*, и присвойте параметрам начальные значения, которые удовлетворят потребности большинства пользователей.

Для этого вам нужно:

- определить значения по умолчанию для всех параметров настройки;
- предусмотреть возможность чтения и изменения значений параметров;
- немедленно применять любые изменения;
- реализовать поддержку активного и неактивного состояний расширения.

8.4.1. Значения настроек по умолчанию

Инфраструктура поддержки виджетов автоматически устанавливает начальные значения параметров настройки, копируя их в атрибут `options` экземпляра объекта виджета, обычно доступный через ссылку `this`. Копируются все значения по умолчанию, указанные в объявлении виджета, и любые значения, передаваемые в вызов функции инициализации.

В листинге 8.5 показано, как определяются значения по умолчанию для параметров настройки. Напомню, что часть этого объекта формируется из определений в объявлении виджета.

Листинг 8.5. Значения по умолчанию параметров настройки расширения

```
// ❶ Глобальные значения по умолчанию параметров для maxlength
options: {
    max: 200,       // Максимальная длина
    truncate: true, // true - запретить ввод сверх ограничения,
        // false - только сообщить о превышении
    showFeedback: true,   // true: всегда отображать подсказку,
        // 'active' - отображать только при наведении указателя мыши
        // или при получении фокуса ввода
    feedbackTarget: null, // селектор jQuery или функция
        // для получения элемента, где должна выводиться подсказка
    feedbackText: '{r} characters remaining ({m} maximum)',
        // Отображаемый текст подсказки,
        // используйте {r} для вывода числа оставшихся (remaining)
        // символов, {c} - для вывода числа уже введенных символов,
        // {m} - для вывода максимально допустимого значения
    overflowText: '{o} characters too many ({m} maximum)',
        // Текст, отображаемый по достижении максимума,
        // используйте символы подстановки, описанные выше,
        // а также {o} - для вывода числа лишних символов
    full: null // Функция, которая вызывается при заполнении
        // или переполнении, принимает два параметра:
        // событие и объект с атрибутом overflow, установленным
        // в значение true, если произошло переполнение,
        // и в значение false в противном случае
},
```

Значения по умолчанию определяются в атрибуте прототипа виджета ❶. Чтобы упростить доступ к ним, вы можете отобразить этот атрибут непосредственно в множество значений по умолчанию:

```
// Упростить доступ к настройкам
$.kbw.maxlength.options = $.kbw.maxlength.prototype.options;
```

После этого можно будет переопределять значения по умолчанию для всех экземпляров расширения, как показано ниже, до применения расширения к каким-либо элементам:

```
$.extend($.kbw.maxlength.options, {max: 300, truncate: false});
```

8.4.2. Реакция на изменение параметров

Инфраструктура поддержки виджетов также автоматически обрабатывает попытки извлечения и изменения значений параметров настройки расширения. При использовании в режиме чтения метод `option` позволяет определить имя желаемого параметра и возвращает его текущее значение (учитывая значения по умолчанию). Если метод вызывается без аргументов, возвращается полный набор параметров настройки:

```
var maxChars = $('#text1').maxlength('option', 'max');
var options = $('#text1').maxlength('option');
```

При использовании в режиме записи метод `option` может изменить значение единственного параметра, если передать ему имя параметра и новое его значение, или группы параметров, если передать объект с именами и значениями параметров:

```
$('#text1').maxlength('option', 'max', 400);
$('#text1').maxlength('option', {max: 400, truncate: false});
```

Выбор того или иного режима определяется количеством и типами аргументов.

Имеется возможность внедрить свои операции в процесс установки новых значений, исследовать изменения и реагировать на них. Для этого достаточно определить в виджете собственные функции `_setOption` и `_setOptions`. В листинге 8.6 демонстрируется пример реализации обработки события изменения единственного параметра.

Листинг 8.6. Обработка события изменения единственного параметра

```
/* Обработка события изменения параметра настройки.
   @param  key    (string)  имя изменяемого параметра настройки
   @param  value  (любой)   новое значение */
// ❶ Переопределить функцию _setOption
_setOption: function(key, value) {
    // ❷ Выполнить операцию для конкретного параметра настройки
    switch (key) {
        case 'disabled':
            this.element.prop('disabled', value);
            this.feedbackTarget.toggleClass('ui-state-disabled', value);
```

```
            break;
    }
    // ❸ Вызвать базовый метод обработки этого события
    this._superApply(arguments);
},
```

Функция _setOption ❶ принимает два аргумента: имя параметра настройки и новое значение. Если в одном вызове передать ей несколько параметров настройки, они будут обработаны по отдельности. В зависимости от имени параметра ❷ функция решает, какие действия следует выполнить. В расширении MaxLength необходимо следить за изменениями параметра disabled и изменять его внешний вид в соответствии со значением.

Не забудьте вызвать базовый метод виджета, реализующий обработку события изменения значения параметра ❸, чтобы он смог обновить значение, хранящееся в экземпляре объекта виджета. До вызова этого метода вы имеете доступ к старому (this.options[key]) и новому (value) значениям параметра и можете сравнить их между собой.

Как упоминалось выше, функция _setOption вызывается отдельно для каждого изменяемого параметра. Если потребуется выполнить некоторые предварительные операции до и после всех изменений, определите функцию _setOptions, как показано в листинге 8.7.

Листинг 8.7. Обработка события изменения нескольких параметров

```
/* Обработка события изменения нескольких параметров настройки.
   @param  option  (object) новые значения параметров */
// ❶ Переопределить функцию _setOptions
_setOptions: function(options) {
    // ❷ Вызвать базовый метод обработки этого события
    this._superApply(arguments);
    // ❸ Перерисовать виджет
    this.refresh();
},
```

Функция _setOptions ❶ принимает единственный аргумент: объект с именами и новыми значениями параметров настройки. Вы всегда должны вызывать стандартную реализацию обработки события изменения параметров, в ходе которой выполняется только что описанная обработка событий изменения отдельных параметров ❷. Затем можно перерисовать виджет, чтобы учесть новые значения параметров ❸. Благодаря тому что виджет перерисовывается в этой функции, а не в предыдущей, снижаются накладные расходы, отрицательно сказывающиеся на производительности, потому что в таком случае данная операция будет выполняться не для каждого изменения в отдельно-

сти, а сразу для целой группы изменений. Функция refresh описывается в следующем разделе.

Версии jQuery UI ниже 1.9

В версии jQuery UI 1.9 изменился способ вызова унаследованных функций. В новых версиях jQuery UI такие функции должны вызываться через ссылку на родительский класс, как показано ниже:

```
this._superApply(arguments);
```

В старых версиях вызов унаследованных функций должен производиться через прототип виджета, например так:

```
$.Widget.prototype._setOption.apply(this, arguments);
```

или так:

```
$.Widget.prototype._setOptions.apply(this, arguments);
```

Чтобы виджет мог работать и со старыми, и с новыми версиями jQuery UI, можно проверить наличие новой функции и использовать соответствующий способ вызова:

```
// Вызвать базовый метод виджета
if (this._superApply) {
    this._superApply(arguments);
}
else {
    $.Widget.prototype._setOption.apply(this, arguments);
}
```

8.4.3. Реализация параметров настройки MaxLength

С помощью параметров настройки можно влиять на внешний вид и поведение расширения MaxLength. На рис. 8.4 изображено действие изменения некоторых параметров.

Значения параметров необходимо заново применять к целевым элементам по мере их изменения и, возможно, устранять последствия действия предыдущих значений. Эту задачу решает собственная функция refresh. Она вызывается при инициализации (см. раздел 8.3.1) и при изменении значений параметров (см. раздел 8.4.2). Первая половина этой функции представлена в листинге 8.8.

Листинг 8.8. Обновление внешнего вида и поведения виджета

```
/* Перерисовывает виджет maxlength. */
// ❶ Определение функции refresh
refresh: function() {
    // ❷ Если имеется элемент для вывода подсказки...
    if (this.feedbackTarget.length > 0) {
        // Удалить старый элемент с текстом подсказки
```

```
        // ❸ ...сбросить в исходное состояние внешний элемент,
        //       если был указан
        if (this.hadFeedbackTarget) {
            this.feedbackTarget.empty().val('').
                removeClass((this.widgetFullName ||
                    this.widgetBaseClass) + this._feedbackClass + ' ' +
                    this._fullClass + ' ' + this._overflowClass);
        }
        else {
            // ❹ Удалить встроенный элемент с подсказкой
            this.feedbackTarget.remove();
        }
        // ❺ Сбросить ссылку на элемент с подсказкой
        this.feedbackTarget = $([]);
    }
    // ❻ Если включен параметр, требующий выводить подсказку,
    // выполнить пункты 7-11
    if (this.options.showFeedback) { // Добавить новый элемент с подсказкой
        // ❼ Запомнить, был ли назначен для вывода подсказки
        //    внешний элемент
        this.hadFeedbackTarget = !!this.options.feedbackTarget;
        // ❽ Вызвать функцию, возвращающую ссылку на внешний
        //    элемент с подсказкой, если это - функция
        if ($.isFunction(this.options.feedbackTarget)) {
            this.feedbackTarget =
                this.options.feedbackTarget.apply(this.element[0],[]);
        }
        // ❾ Иначе отыскать внешний элемент
        else if (this.options.feedbackTarget) {
            this.feedbackTarget = $(this.options.feedbackTarget);
        }
        // ❿ Создать встроенный элемент с подсказкой
        else {
            this.feedbackTarget =
                $('<span></span>').insertAfter(this.element);
        }
        // ⓫ Настроить элемент с подсказкой
        this.feedbackTarget.addClass(
            (this.widgetFullName || this.widgetBaseClass) +
            this._feedbackClass + ' ui-state-default' +
            (this.options.disabled ? ' ui-state-disabled' : ''));
    }
    ...
},
```

Функция `refresh` ❶ переопределяет унаследованную версию. Функции, имена которых не начинаются с символа подчеркивания (_), допускается вызывать непосредственно как методы:

```
$('#text1').maxlength('refresh');
```

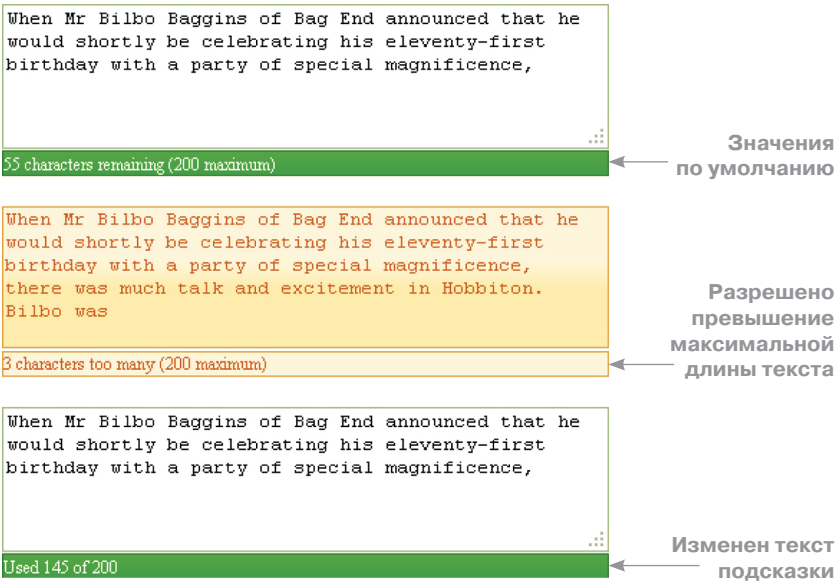


Рис. 8.4 ❖ Влияние изменения параметров настройки MaxLength (сверху вниз): значения по умолчанию, без усечения, изменение текста подсказки

Из-за того, что параметры настройки вывода подсказки могут изменяться, сначала необходимо удалить элемент с подсказкой. После проверки наличия элемента с подсказкой, которая сводится к проверке ссылки ❷, этот элемент очищается и отключается от виджета, если он – внешний ❸, и удаляется полностью, если он – встроенный ❹. В обоих случаях сбрасывается ссылка на старый элемент ❺.

Затем проверяется параметр `showFeedback` ❻, чтобы выяснить, требуется ли выводить подсказку. Если он имеет истинное значение, необходимо запомнить, был ли указан для вывода подсказки внешний элемент ❼, чтобы в будущем его можно было корректно сбросить в исходное состояние. Если внешний элемент подсказки определен как функция ❽, необходимо вызвать ее, чтобы получить ссылку на фактический элемент. В противном случае ссылка на внешний элемент подсказки извлекается с помощью jQuery ❾, что дает возможность определять внешний элемент в виде строки селектора, ссылки на элемент DOM или существующий объект jQuery. Если внешний элемент не указан в настройках, необходимо создать собственный и добавить его после элемента `textarea` ❿. В любом случае ссылка на

элемент с подсказкой сохраняется в атрибуте `feedbackTarget` экземпляра объекта (`this`) для дальнейшего использования, а в сам элемент добавляются соответствующие классы оформления ThemeRoller ❶❶, чтобы обеспечить единообразие оформления с остальной страницей.

Действие конструкции `!!` объясняется в разделе 3.2.1.

Вторая половина функции `refresh`, представленная в листинге 8.9, настраивает вывод подсказки, если пользователь затребовал выводить ее, только при наведении указателя мыши или получении элементом фокуса ввода.

Листинг 8.9. Установка обработчиков событий в функции refresh

```
/* Перерисовывает виджет maxlength. */
refresh: function() {
    ...
    // ❶ Удалить прежние обработчики событий
    this.element.unbind('mouseover.' + this.widgetEventPrefix +
        ' focus.' + this.widgetEventPrefix +
        ' mouseout.' + this.widgetEventPrefix +
        ' blur.' + this.widgetEventPrefix);
    // ❷ Если подсказка должна отображаться только при
    //    наведении указателя мыши или получении фокуса ввода,
    //    выполнить пункты 3-6
    if (this.options.showFeedback == 'active') {
        // Дополнительные обработчики событий
        var self = this;
        this.element.
            // ❸ Выводить подсказку при наведении указателя мыши
            bind('mouseover.' + this.widgetEventPrefix, function() {
                self.feedbackTarget.css('visibility', 'visible');
            // ❹ Скрывать подсказку, когда указатель мыши выходит
            //    за границы элемента
            }).bind('mouseout.' + this.widgetEventPrefix, function() {
                if (!self.focussed) {
                    self.feedbackTarget.css('visibility', 'hidden');
                }
            // ❺ Выводить подсказку при получении фокуса ввода
            }).bind('focus.' + this.widgetEventPrefix, function() {
                self.focussed = true;
                self.feedbackTarget.css('visibility', 'visible');
            // ❻ Скрывать подсказку при потере фокуса ввода
            }).bind('blur.' + this.widgetEventPrefix, function() {
                self.focussed = false;
                self.feedbackTarget.css('visibility', 'hidden');
            });
        // ❼ Изначально подсказка должна быть скрыта
        this.feedbackTarget.css('visibility', 'hidden');
```

```
    }
    // ❽ Проверить длину текста в элементе ввода
    this._checkLength();
},
```

Как и в случае с элементом, предназначенным для вывода подсказки (см. листинг 8.8), сначала необходимо удалить все обработчики событий, отвечавшие за вывод подсказки ❶. Использование пространства имен при назначении обработчиков любых событий позволяет упростить функцию и удалять обработчики, не опасаясь конфликтов с обработчиками, возможно установленными другими библиотеками. Здесь нельзя просто удалить сразу все обработчики, принадлежащие данному пространству имен, – обработчики событий нажатий на клавиши, созданные на этапе инициализации, должны оставаться нетронутыми.

Если пользователь потребовал выводить подсказку только при наведении указателя мыши или когда элемент владеет фокусом ввода ❷, необходимо подключить соответствующие обработчики событий. Подсказка должна отображаться при наведении указателя мыши на элемент `textarea` ❸ и скрываться, когда он выходит за границы элемента ❹, но только если элемент не владеет фокусом ввода. Аналогично подсказка должна отображаться при получении фокуса ввода элементом `textarea` ❺ и скрываться, когда он теряет фокус ввода ❻. Как отмечалось выше, имена событий должны включать название пространства имен, хранящееся в атрибуте `widgetEventPrefix` и являющееся уникальным для каждого виджета, чтобы упростить удаление обработчиков в будущем. В любом случае изменяется свойство `visibility` элемента подсказки, благодаря чему изменяется только видимость элемента, а пространство, занимаемое им на странице, резервируется всегда. Первоначально элемент подсказки должен быть скрыт ❼.

Наконец необходимо применить ограничения на максимальную длину текста в элементе `textarea`, вызвав функцию `_checkLength` ❽. Подробное описание этой функции приводится в разделе 8.8.1.

8.4.4. Активация и деактивация виджета

Активация (enabling) и деактивация (disabling) виджета являются стандартной функциональностью, предоставляемой инфраструктурой, и не требует писать дополнительный код.

Состояние виджета контролируется параметром `disabled`, который может принимать значения `true` (неактивное состояние) и `false` (ак-

тивное состояние). Кроме того, инфраструктура поддерживает методы enable и disable, изменяющие значение этого параметра:

```
$('#text1').maxlength('disable');
...
$('#text1').maxlength('enable');
```

Внутри инфраструктура переключает два класса в главном элементе: <пространство_имен>-<имя_расширения>-disabled и ui-state-disabled – и устанавливает соответствующее значение в атрибуте aria-disabled. При желании можно предусмотреть выполнение собственных операций в ответ на изменение состояния элемента, анализируя значение параметра disabled в функции _setOption (как показано в разделе 8.4.2).

Теперь, когда пользователь получил возможность настраивать ваш новый виджет, давайте посмотрим, как они смогут обрабатывать изменение состояния с помощью обработчиков событий.

8.5. Добавление обработчиков событий

Реализовав поддержку событий, вы можете дать пользователям возможность выполнять дополнительные операции в ответ на наиболее важные события, происходящие внутри вашего расширения. Иногда в процессе работы расширения могут возникать ситуации, о которых пользователи могут пожелать узнавать немедленно. Примером таких ситуаций может служить событие change, возникающее при каждом изменении содержимого поля ввода. Получая такое событие, пользователи смогут немедленно обновлять страницу в соответствии с новым состоянием расширения. Инфраструктура поддержки виджетов имеет возможность возбуждения событий посредством функции _trigger.

Чтобы подключить обработчик события к расширению, необходимо:

- дать пользователям возможность регистрировать свои обработчики событий;
- возбуждать события в соответствующие моменты времени.

Давайте посмотрим, как это реализуется на практике.

8.5.1. Регистрация обработчиков событий

Данным расширением поддерживается только одно событие: full. Оно возникает, когда длина текста в элементе textarea достигает или превышает установленное максимально допустимое число символов.

В качестве одного из параметров обработчику события передается флаг, который указывает, было ли превышено максимально допустимое число символов (это может произойти, если параметр настройки `truncate` установлен в значение `false`) и требуется ли сократить длину текста перед отправкой на сервер.

Ссылка на обработчик `full` – это всего лишь еще один параметр настройки расширения, как показано в листинге 8.10, по умолчанию имеющий значение `null`. Чтобы подключить свой обработчик события, пользователь должен присвоить этому параметру ссылку на свою функцию, принимающую два аргумента, как и другие обработчики событий в jQuery UI. В первом аргументе передается само событие, а во втором – объект `ui`, хранящий флаг переполнения `overflow`. Внутри обработчика переменная `this` ссылается на элемент `textarea`, к которому подключено расширение, что дает пользователю возможность использовать одну и ту же функцию-обработчик для обслуживания событий сразу от нескольких экземпляров расширения.

Листинг 8.10. Определение обработчика событий

```
// Настройки виджета по умолчанию
options: {
    ...
    full: null // Функция, которая вызывается при заполнении
        // или переполнении, принимает два параметра:
        // событие и объект с атрибутом overflow, установленным
        // в значение true, если произошло переполнение,
        // и в значение false в противном случае
},
```

Пользователи могут зарегистрировать свой обработчик событий в процессе инициализации расширения, как показано в следующем фрагменте, или изменяя параметры настройки позднее:

```
$('#text1').maxlength({full: function(event, ui) {
    $('#warning').show();
}});
```

Инфраструктура поддержки виджетов способна автоматически вызывать обработчики событий, благодаря чему для подписки на события можно использовать стандартные функции `bind` и `on` из библиотеки jQuery. Имя события определяется как комбинация из имени расширения и имени параметра (в данном случае `maxlengthfull`).

```
$('#text1').maxlength().on('maxlengthfull', function(event, ui) {
    $('#warning').show();
}});
```

После установки обработчика событий необходимо обеспечить его вызов в соответствующие моменты времени.

8.5.2. Вызов обработчиков событий

Событие full возбуждается в расширении MaxLength в конце функции _checkLength сразу после проверки ограничений, как показано в листинге 8.11. В других расширениях события могут возбуждаться в иные моменты времени, наиболее подходящие для этого.

Листинг 8.11. Возбуждение события

```
/* Проверяет длину текста и при необходимости вызывает обработчик. */
_checkLength: function() {
    var value = this.element.val();
    var len = value.replace(/\r\n/g, '~~').replace(/\n/g, '~~').length;
    ...
    // ❶ Проверить необходимость возбуждения события
    if (len >= this.options.max) {
        // ❷ Возбудить событие
        this._trigger('full', null, $.extend(this.curLength(),
           {overflow: len > this.options.max}));
    }
},
```

Сначала проверяется условие возбуждения события ❶, затем, при необходимости, вызывается функция _trigger виджета ❷. В качестве аргументов ей передаются: имя события, оригинальное событие, вызвавшее данное событие, и объект ui с информацией, определяемой самим виджетом. Передача значения null в качестве оригинального события вынуждает jQuery создать объект события с типом, определяемым именем расширения и именем события: maxlengthfull. Имейте в виду: если имя события совпадает с именем виджета, одно и то же имя не будет дублироваться при создании имени типа события – будет использоваться только одно имя. Например, для события drag в расширении с именем drag будет создан объект события с типом drag.

При желании можно дать пользователю возможность отменить выполнение действий по умолчанию, предусмотренных виджетом, через вызов метода event.preventDefault() в обработчике событий. Это заставит функцию _trigger вернуть false. Вы можете проверить возвращаемое значение и предусмотреть соответствующую реакцию на него. Например, возбуждение события в расширении могло бы выглядеть так:

```
if (this._trigger('myevent', null, {value: this.value})) {
    ... // Только если действие по умолчанию не было отменено
}
```

В этом случае пользователь смог бы отменить действия по умолчанию, если атрибут `value` имеет значение больше 10:

```
$(selector).myplugin({myevent: function(event, ui) {
        if (ui.value > 10) {
            event.preventDefault();
        }
    }
});
```

Поддержка событий позволяет организовать выполнение операций в ответ на события, происходящие внутри самого виджета. Чтобы пользователь мог самостоятельно инициировать выполнение виджетом каких-либо операций, необходимо добавить дополнительные методы, чтобы их можно было вызывать по мере необходимости, передавая имя метода и какие-либо параметры для него.

8.6. Добавление методов

Для реализации других функциональных возможностей внутри расширения используются методы. Вызов метода осуществляется обращением к функции виджета и передачей ей имени метода, как показано ниже:

```
$('#text1').maxlength('enable');
$('#text1').maxlength('disable');
```

Инфраструктура поддержки виджетов обеспечивает такую возможность, позволяя вызывать как методы любые функции внутри виджета, имена которых *не* начинаются с символа подчеркивания (_). Если такая функция возвращает некоторое значение, оно будет передано вызывающему коду. В противном случае инфраструктура вернет текущую коллекцию jQuery, чтобы дать возможность продолжить цепочку вызовов.

Любые функции с именами, начинающимися с символа подчеркивания, считаются внутренними и недоступны как методы. Но к ним можно обратиться непосредственно через экземпляр объекта виджета. Например, начиная с версии jQuery UI 1.9 допускается выполнять такие вызовы:

```
$('#text1').data('kbw-maxlength')._setOptions({...});
```

а в версиях jQuery UI ниже 1.9:

```
$('#text1').data('maxlength')._setOptions({...});
```

8.6.1. Получение текущей длины

Как пример реализации метода предоставим пользователю возможность получать текущее количество символов обращением к методу `curLength`, представленному в листинге 8.12. Он возвращает объект с атрибутами `used` (введенное количество символов) и `remaining` (количество символов, оставшееся до достижения максимально допустимой длины). Обратите внимание, что если настройки виджета допускают возможность переполнения, значение `used` может оказаться больше значения параметра `max`, а атрибут `remaining` может иметь отрицательное значение.

Листинг 8.12. Извлечение текущей длины

```
/* Возвращает число символов, введенных и оставшихся до предельного значения.
   @return  (object) текущие счетчики символов в виде атрибутов
                     used и remaining */
// ❶ Функция, реализующая метод curLength
curLength: function() {
    var value = this.element.val();
    var len = value.replace(/\r\n/g, '~~').replace(/\n/g, '~~').length;
    // ❷ Вернуть текущие значения длины
    return {used: len, remaining: this.options.max - len};
},
```

Чтобы связать метод с функцией реализации, необходимо дать этой функции имя, совпадающее с именем метода ❶. Функция должна возвращать желаемое значение ❷, которое будет передано непосредственно в вызывающий код. Если функция ничего не возвращает, инфраструктура поддержки виджетов автоматически вернет оригинальный объект jQuery, чтобы дать возможность включать обращения к этому методу в цепочки вызовов.

Данный метод вызывается, как показано ниже:

```
var lengths = $('#text1').maxlength('curLength');
alert(lengths.remaining + ' characters remaining');
```

Методы в других виджетах могут позволять или требовать передачу аргументов, управляющих их поведением, как, например, метод установки бегунка в новую позицию:

```
$('#slider').slider('value', 25);
```

Чтобы реализовать передачу аргументов, необходимо перечислить их в объявлении функции, как обычно, и использовать в теле самой функции, а инфраструктура автоматически будет передавать любые дополнительные значения, указанные в обращении к методу.

Код, который мы видели до сих пор, позволяет подключать виджет MaxLength к указанным элементам, читать и изменять параметры настройки, а также регистрировать обработчики событий. Но пользователю может также потребоваться удалить виджет в какой-то момент времени.

8.7. Удаление виджета

Чтобы удалить все следы пребывания виджета, пользователи вызывают метод `destroy`. Как и в случае с другими методами, обращение к нему отображается в вызов функции с тем же именем, объявленной внутри расширения.

8.7.1. Метод _destroy

Чтобы реализовать собственный метод `destroy`, необходимо переопределить унаследованную функцию `_destroy`. Встроенная функция `destroy` автоматически вызывает ее для выполнения операций, специфичных для данного виджета. Внутри функции `_destroy` можно отменить все, что было сделано в функциях `_create` и `refresh`, как показано в листинге 8.13.

Листинг 8.13. Удаление виджета

```
/* Удаляет функциональные возможности расширения maxlength из элемента */
// ❶ Определение функции, реализующей метод destroy
_destroy: function() {
    // ❷ Если включен вывод подсказки, выполнить пункты 3-5
    if (this.feedbackTarget.length > 0) {
        if (this.hadFeedbackTarget) {
            // ❸ Сбросить внешний элемент в исходное состояние
            this.feedbackTarget.empty().val('').
                css('visibility', 'visible').
                removeClass((this.widgetFullName ||
                    this.widgetBaseClass) +
                    this._feedbackClass + ' ' +
                    this._fullClass + ' ' +
                    this._overflowClass);
        }
        else {
            // ❹ Удалить встроенный элемент подсказки
            this.feedbackTarget.remove();
        }
    }
    // ❺ Удалить функциональность расширения
```

```
    this.element.removeClass(
            (this.widgetFullName || this.widgetBaseClass) + ' ' +
            this._fullClass + ' ' + this._overflowClass).
        unbind('.' + this.widgetEventPrefix);
}
```

Здесь объявляется функция, переопределяющая встроенный метод `destroy` ❶. В расширении MaxLength сначала необходимо узнать, отображалась ли подсказка, для чего выполняется проверка атрибута, ссылающегося на элемент с подсказкой ❷. Если подсказка отображалась, следующим шагом следует определить, является ли элемент подсказки внешним, и в этом случае сбросить его в исходное состояние ❸, или внутренним, и тогда просто удалить его ❹.

Затем нужно удалить класс-метку и любые другие классы, добавленные виджетом, а также все подключенные обработчики событий ❺. Удаление обработчиков существенно упрощается благодаря использованию пространства имен при их добавлении. Достаточно просто передать функции `unbind` название пространства имен, чтобы безопасно удалить все обработчики имен, установленные виджетом, и оставить все остальные нетронутыми.

После вызова этого метода из выбранных элементов удаляются функциональные возможности расширения MaxLength, и они возвращаются в исходное состояние.

Версии jQuery UI ниже 1.9

В версиях jQuery UI ниже 1.9 вместо функции `_destroy` необходимо переопределять функцию `destroy` и вызывать унаследованную функцию через прототип виджета. Чтобы виджет мог работать и со старыми, и с новыми версиями jQuery UI, в старых версиях необходимо переопределить функцию `destroy` и заставить ее вызывать функцию `_destroy`, как это делается в последних версиях. Может также потребоваться вызывать унаследованную функцию `destroy`, реализованную в инфраструктуре поддержки виджетов. Как это делается, показано ниже:

```
if (!$.Widget.prototype._destroy) {
    $.extend(maxlengthOverrides, {
        /* Удалить функциональные возможности
           расширения maxlength из элемента. */
        destroy: function() {
            this._destroy();
            // Вызвать базовый метод виджета
            $.Widget.prototype.destroy.call(this);
        }
    });
}
```

8.8. Заключительные штрихи

На этом мы заканчиваем обзор взаимодействий между расширением и инфраструктурой поддержки виджетов. Но мы можем внести еще пару заключительных штрихов в обсуждаемое расширение:

- реализация главной цели расширения;
- поддержка стилей оформления, определяющих внешний вид.

Главная цель расширения MaxLength – ограничить длину текста, который можно ввести в элемент `textarea`.

8.8.1. Главная цель расширения

Расширение MaxLength проектировалось с целью дать возможность ограничивать длину текста, который можно ввести в элемент `textarea`. До сих пор мы рассматривали особенности интеграции расширения с инфраструктурой поддержки виджетов и настройки стандартной функциональности. Как вы могли убедиться выше, в конечном итоге все пути сходятся в функции `_checkLength`, выполняющей основную работу по определению текущей длины содержимого и его усечению, если необходимо. Реализация этой функции представлена в листинге 8.14.

Листинг 8.14. Проверка длины содержимого в поле ввода

```
/* Проверяет длину текста и выводит соответствующую подсказку. */
_checkLength: function() {
    var value = this.element.val();
    // ❶ Нормализовать окончания строк
    var len = value.replace(/\r\n/g, '~~').replace(/\n/g, '~~').length;
    // ❷ Установить текущее состояние элемента textarea
    this.element.toggleClass(this._fullClass, len >= this.options.max).
        toggleClass(this._overflowClass, len > this.options.max);
    // ❸ Выполнить усечение текста
    if (len > this.options.max && this.options.truncate) {
        // Выполнить усечение
        var lines = this.element.val().split(/\r\n|\n/);
        value = '';
        var i = 0;
        while (value.length < this.options.max && i < lines.length) {
            value += lines[i].substring(
                0, this.options.max - value.length) + '\r\n';
            i++;
        }
        this.element.val(value.substring(0, this.options.max));
        this.element[0].scrollTop =
            this.element[0].scrollHeight; // Прокрутить вниз
```

```
        len = this.options.max;
    }
    // ❹ Установить текущее состояние подсказки
    this.feedbackTarget.toggleClass(
            this._fullClass, len >= this.options.max).
        toggleClass(this._overflowClass, len > this.options.max);
    // ❺ Сгенерировать и вывести текст подсказки
    var feedback = (len > this.options.max ? // Подсказка
        this.options.overflowText : this.options.feedbackText).
            replace(/\{c\}/, len).
            replace(/\{m\}/, this.options.max).
            replace(/\{r\}/, this.options.max - len).
            replace(/\{o\}/, len - this.options.max);
    try {
        this.feedbackTarget.text(feedback);
    }
    catch(e) {
        // Игнорировать
    }
    try {
        this.feedbackTarget.val(feedback);
    }
    catch(e) {
        // Игнорировать
    }
    // ❻ Вызвать обработчик события, если необходимо
    if (len >= this.options.max) {
        this._trigger('full', null, $.extend(this.curLength(),
            {overflow: len > this.options.max}));
    }
},
```

Сначала функция определяет текущую длину текста, учитывая различия между браузерами в оформлении окончаний строк ❶. Опираясь на это значение и параметры настройки расширения, к элементу `textarea` применяется один или два класса, чтобы отразить его текущее состояние – заполнен или переполнен ❷. Напомню, что переменная `this` ссылается на текущий экземпляр объекта виджета, основными атрибутами которого являются ссылка на элемент (`element`) и массив с текущими параметрами настройки (`options`).

Если длина текста превышает установленный максимум и пользователь потребовал усекать лишний текст ❸, вычисляется усеченное значение, при этом снова выполняется нормализация окончаний строк. Когда текст будет записан обратно в элемент `textarea`, он автоматически прокрутит содержимое вверх. Однако обычно текст вводится в конце, поэтому далее выполняется прокрутка вниз, чтобы по-

мочь пользователю. Так как было произведено усечение текста, текст подсказки должен измениться, чтобы отразить этот факт.

Далее, исходя из новой длины текста, к элементу подсказки применяется один или два класса ❹. Так как длина текста может уменьшиться из-за его усечения, между элементами `textarea` и подсказки могут появиться расхождения. Это сделано преднамеренно, так как подсказка должна отражать обновленное состояние, а элемент `textarea` своим внешним видом должен показывать, что предпринята неудачная попытка ввести дополнительный текст.

Исходя из состояния элемента, выводится сообщение в указанном элементе подсказки ❺. Для большей гибкости в качестве элемента подсказки расширение допускает использовать `div`, `span`, `p` или `input`. Но текст в эти элементы записывается по-разному, и использование неправильного способа может вызвать исключение в некоторых браузерах. Поэтому запись текста подсказки выполняется здесь внутри двух инструкций `try/catch`.

Наконец, чтобы сообщить о наполнении или переполнении элемента, вызывается обработчик `full` ❻. Тема вызова обработчика события подробно рассматривалась в разделе 8.5.2.

На этом мы заканчиваем знакомство с функциональными возможностями расширения. Следующий шаг – реализация поддержки стилей оформления, чтобы обеспечить единство внешнего вида с другими виджетами на странице.

8.8.2. Реализация поддержки стилей

Для придания единого внешнего вида всем виджетам на странице библиотека jQuery UI использует инструмент ThemeRoller, с помощью которого генерирует таблицы стилей CSS для своих виджетов, придает единый стиль оформления всем компонентам и поддерживает принцип *оформления расширений с помощью CSS*. Внешний вид виджетов в значительной степени обусловлен применением к элементам стандартных классов, ссылающихся на стили в ThemeRoller. Если вам потребуется реализовать какое-то дополнительное оформление, создайте внешний файл CSS с именем, совпадающим с именем файла, хранящим программный код, но с другим расширением. Например, как в данном случае: `jquery.maxlengthui.css`.

Программный код расширения присваивает несколько классов элементам, которыми он управляет. Эти классы обеспечивают единство оформления расширения и других компонентов jQuery UI на странице. Элемент подсказки первоначально включает класс

ui-state-default. При наполнении или переполнении ему и элементу ввода textarea присваиваются классы ui-state-highlight и ui-state-error соответственно. Когда виджет переключается в неактивное состояние, к обоим элементам – textarea и к элементу подсказки – применяется стандартный класс ui-state-disabled.

Благодаря использованию стандартных классов из ThemeRoller потребность в специфических стилях CSS сводится к минимуму и касается только оформления элементов подсказки, как показано в листинге 8.15.

Листинг 8.15. Стили CSS для виджета

```
/* Стили CSS для виджета MaxLength v1.0.0 */

/* Стиль оформления элемента подсказки */
.kbw-maxlength-feedback {
    margin: 0em 0em 0em 0.5em;
    font-size: 75%;
    font-weight: normal;
}
```

Написав всего несколько строк кода CSS, пользователь сможет изменить внешний вид виджета, как показано на рис. 8.5.

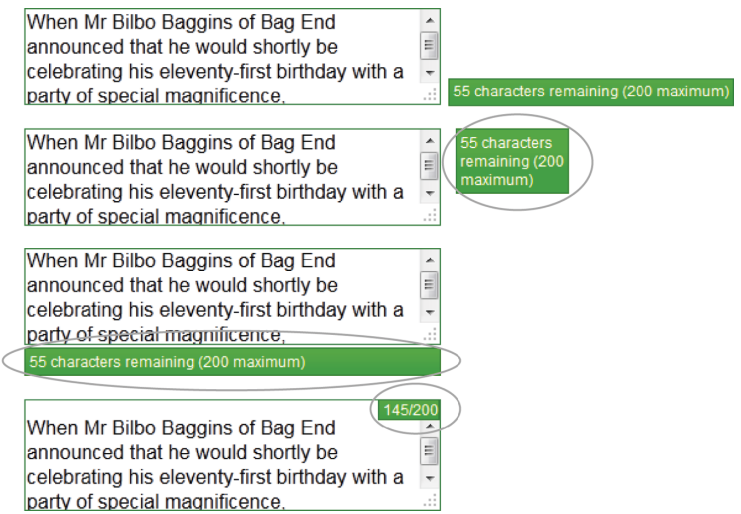


Рис. 8.5 ❖ Оформление расширения MaxLength (сверху вниз): стиль по умолчанию, компактный стиль, подсказка под элементом ввода и подсказка поверх элемента ввода

8.9. Законченное расширение

Итак, мы закончили разработку виджета для jQuery UI, ограничивающего длину текста в поле ввода `textarea`. Инфраструктура поддержки виджетов, входящая в состав библиотеки jQuery UI, упрощает создание расширений и помогает следовать принципам проектирования, описанным в главе 4. Полный код расширения доступен для загрузки на сайте книги.

Дополнительные функциональные возможности виджетов

Инфраструктура поддержки виджетов обладает рядом дополнительных возможностей, не затронутых в этой главе.

Константы

При создании расширения MaxLength мы использовали лишь две константы из нескольких, определяемых инфраструктурой. Все они доступны внутри функций через экземпляр объекта виджета (`this`).

Константа	Назначение	Пример
`namespace`	Пространство имен, указанное в определении виджета	`kbw`
`widgetBaseClass`	Имя класса, используемое как префикс или как самостоятельное имя (в версиях jQuery UI ниже 1.9)	`kbw-maxlength`
`widgetEventPrefix`	Имя, используемое как префикс в именах событий	`maxlength`
`widgetFullName`	Имя класса, используемое как префикс или как самостоятельное имя (в версиях jQuery UI 1.9 и выше)	`kbw-maxlength`
`widgetName`	Имя расширения, указанное в его определении	`maxlength`

Событие create

Еще один способ внедриться в жизненный цикл виджета – реализовать обработку события `create`, которое генерируется в процессе инициализации. Это событие возбуждается сразу после возврата из функции `_create` и дает возможность внести дополнительные изменения сразу, как только виджет будет полностью готов к работе:

```
$('#text1').maxlength({create: function(event, ui) {
    ...
}});
```

Инициализация виджета

Для инициализации конкретного экземпляра расширения мы переопределяли функцию `_create` (см. раздел 8.3.1), однако существует возможность продолжить инициализацию экземпляров виджета после начально-

го этапа создания. Функция `_init` вызывается вслед за функцией `_create` и генерирует событие `create`. Она не принимает аргументов, а переменная `this` в ней ссылается на экземпляр объекта виджета, как обычно.

```
$.widget('kbw.maxlength', {
    _init: function() {
        ...
    }
}
```

Поддержка метаданных

Инфраструктура поддержки виджетов автоматически использует расширение Metadata (https://github.com/jquery-orphans/jquery-metadata), если оно доступно (вплоть до версии jQuery UI 1.9), что дает возможность настраивать элементы изнутри, а не через параметры настройки. Например, можно выполнить инициализацию, получив необходимые настройки из целевого элемента, как показано ниже:

```
$('#text1').maxlength();

<textarea id="text1" rows="5" cols="50"
          class="{maxlength: {max: 300}}"></textarea>
```

Начиная с версии jQuery UI 1.10 расширение Metadata больше не поддерживается по умолчанию. Но эту поддержку легко включить. Для этого достаточно проверить наличие функции `_getCreateOptions` и восстановить ее, если она отсутствует:

```
if ($.Widget.prototype._getCreateOptions === $.noop) {
    $.extend(maxlengthOverrides, {
        /* Восстановить поддержку метаданных. */
        _getCreateOptions: function() {
            return $.metadata && $.metadata.get(
                this.element[0])[this.widgetName];
        }
    });
}
```

Доступ к виджету

Получить ссылку на главный элемент, которым управляет расширение, из-за пределов самого расширения можно с помощью метода `widget`. По умолчанию эта функция возвращает оригинальный элемент, к которому было подключено расширение (`this.element`), но есть возможность изменить поведение метода так, чтобы он возвращал нечто более полезное. Например, расширение Dialog возвращает ссылку на элемент `div`, обертывающий оригинальный элемент:

```
$('#dialog').dialog({open: function(event, ui) {
    var wrapper = $(this).dialog('widget');
    ...
}});
```

Это нужно знать

Создавайте расширения для jQuery UI, когда возникнет необходимость интеграции с другими виджетами jQuery UI.
Библиотека jQuery UI включает инфраструктуру поддержки виджетов, предоставляющую базовую функциональность, необходимую большинству расширений коллекций.
Защищайте свои расширения и предотвращайте конфликты имен с помощью создания областей видимости.
Используйте функцию `$.widget` для определения новых виджетов.
Переопределяйте унаследованные функции для настройки поведения новых виджетов, а именно: `_create`, `_setOptions`, `_setOption` и `_destroy`.
Предоставляйте параметры настройки в своих расширениях, но всегда инициализируйте их осмысленными значениями по умолчанию.
Используйте именованные методы для предоставления дополнительных функциональных возможностей.
Добавляйте соответствующие классы ThemeRoller в свои элементы для поддержки применяемых тем оформления.

Попробуйте сами

Реализуйте повторно расширение Watermark, о котором рассказывалось в главе 5, в виде виджета jQuery UI. Добавьте параметр для идентификации атрибута, откуда будет извлекаться текст для метки, и метод, который будет выполнять очистку метки.

8.10. В заключение

Библиотека jQuery UI – это официальное дополнение к библиотеке jQuery, включающее множество компонентов для организации взаимодействий с веб-страницей. Она содержит вспомогательные функции, низкоуровневые механизмы (например, поддержку буксировки элементов мышью), высокоуровневые компоненты, или виджеты (такие как Tabs и Datepicker), а также множество визуальных эффектов. Поддержка инструмента ThemeRoller позволяет обеспечить единообразие внешнего оформления всех виджетов на странице.

Виджеты jQuery UI реализуются на основе встроенной инфраструктуры поддержки виджетов, что гарантирует единообразие взаимодействий с ними и простоту сопровождения. Данная инфраструктура является аналогом инфраструктуры поддержки расширений для jQuery, представленной в главе 5.

На примере повторной реализации расширения MaxLength с использованием инфраструктуры поддержки виджетов вы познакомились со сходствами и различиями этих двух подходов. Оба позволяют создавать расширения, прекрасно интегрирующиеся с библиотекой jQuery, поддерживающие хранение информации о состоянии каждо-

го экземпляра, взаимодействующие с пользователем и отменяющие любые изменения, произведенные ими. Эти инфраструктуры дают возможность сосредоточиться на создании уникальной функциональности расширения, не отвлекаясь на низкоуровневые операции.

В следующей главе мы познакомимся с еще одним модулем jQuery UI, упрощающим взаимодействие с мышью, и реализуем возможность рисования с его помощью.

Глава 9

Взаимодействия с мышью в jQuery UI

Эта глава охватывает следующие темы:

- модуль Mouse в библиотеке jQuery UI;
- создание расширения для jQuery UI, использующего мышь.

В предыдущей главе мы познакомились с инфраструктурой поддержки виджетов в библиотеке jQuery UI и прошли через процедуру создания виджета с использованием ее возможностей. Вы увидели, какие функциональные возможности предоставляет инфраструктура и как она помогает обеспечить единство внешнего вида и поведения компонентов jQuery UI. Другой важнейшей особенностью пользовательских веб-интерфейсов является поддержка взаимодействий с мышью, которая в стандартной библиотеке jQuery обеспечивается посредством множества обработчиков событий от мыши. Но все эти обработчики действуют на довольно низком уровне, оставляя реализацию высокоуровневых взаимодействий нам.

В свое время разработчикам библиотеки jQuery UI потребовалось реализовать операцию буксировки мышью в нескольких виджетах. С этой целью был создан отдельный модуль, обеспечивающий простой и единообразный интерфейс для взаимодействий с мышью. Модуль jQuery UI Mouse перехватывает низкоуровневые события от мыши и преобразует их в события перемещения указателя, которые вы можете обрабатывать в своих веб-приложениях. Его можно сравнить со спидометром автомобиля, который преобразует частоту вращения колеса в скорость движения машины.

Чтобы поближе познакомиться с возможностями модуля Mouse, мы создадим расширение, позволяющее рисовать линии с помощью мыши и преобразовывать получившийся рисунок в текстовое представление для дальнейшей обработки. Дополнительно виджет будет позволять проверять наличие рисунка, очищать рисунок и сохранять его для повторного отображения позднее.

9.1. Модуль jQuery UI Mouse

Разработчики jQuery UI осознают, что взаимодействия с мышью являются основой для многих компонентов пользовательского интерфейса, и предоставляют прямую поддержку обработки событий от мыши в виде модуля `ui.mouse`. Этот модуль входит в зависимости модулей Draggable, Resizable, Selectable, Sortable и Slider, что наглядно демонстрирует востребованность такой функциональности. Поддержка низкоуровневых взаимодействий с мышью предоставляется стандартной библиотекой jQuery в форме различных событий и их обработчиков, тогда как jQuery UI преобразует эти низкоуровневые события в высокоуровневые механизмы и позволяет настраивать их работу.

9.1.1. Операции буксировки мышью

Модуль Mouse добавляет обертки для некоторых событий от мыши в целевые элементы и преобразует эти низкоуровневые события в высокоуровневые, непосредственно связанные с операцией буксировки. Благодаря этому в виджетах, опирающихся на данный модуль, остается только реализовать желаемую реакцию на движение указателя мыши и не заботиться об определении момента начала буксировки и последующего ее отслеживания.

То есть вместо обработки низкоуровневых событий `mousedown`, `mousemove` и `mouseup` виджет транслирует их в вызовы функций `_mouseCapture`, `_mouseStart`, `_mouseDrag` и `_mouseStop`, как показано на рис. 9.1. Если вам потребуется организовать в своем виджете поддержку буксировки мышью, просто унаследуйте модуль Mouse, переопределите эти функции, и они автоматически будут вызываться в соответствующие моменты времени.

Модуль Mouse также обрабатывает событие перемещения указателя мыши за границы элемента, где была начата процедура буксировки, и продолжает следить за процессом буксировки как обычно. Это не позволяет участвовать в операции буксировки сразу нескольким элементам и предотвращает возможные конфликты между ними.

9.1.2. Параметры настройки, поддерживаемые модулем Mouse

Модуль Mouse поддерживает несколько параметров настройки его поведения. Вы можете определять новые значения этих параметров

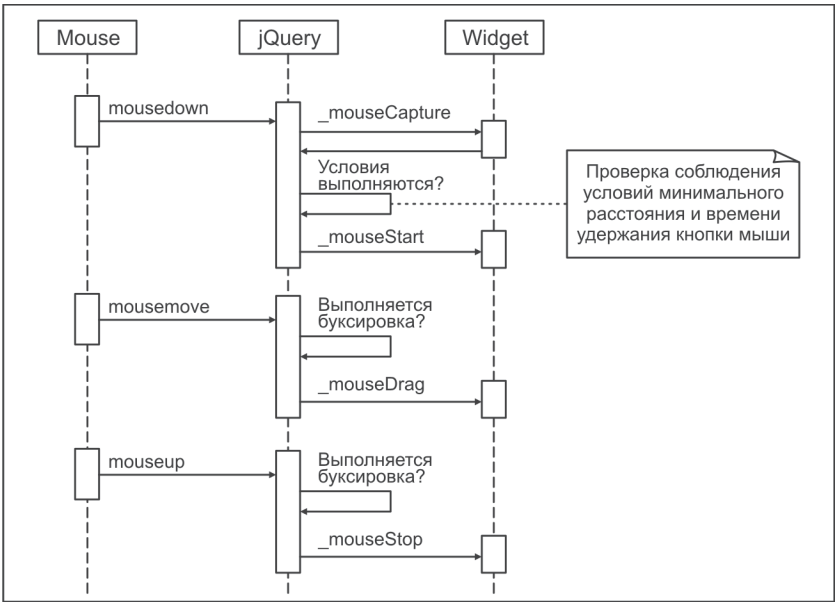


Рис. 9.1 ❖ Схема последовательности событий в виджетах на основе модуля jQuery UI Mouse

в экземплярах виджетов, для чего достаточно просто передать их требуемому экземпляру, как обычно:

```
$('div.item').draggable({distance: 5, delay: 100});
```

Ниже перечислены некоторые из этих параметров.

- `cancel` – определяет селектор, препятствующий запуску последовательности событий буксировки. Селектор применяется к родителям целевого элемента, и при обнаружении совпадения дальнейшая обработка событий прекращается. По умолчанию этот параметр хранит селектор, соответствующий любому полю ввода (включая поля ввода многострочного текста, раскрывающиеся списки и кнопки) или пункту раскрывающегося списка: `:input,option`;
- `distance` – определяет минимальное расстояние, на которое должен переместиться указатель мыши, чтобы была инициирована операция буксировки. Значение по умолчанию – один пиксель;
- `delay` – определяет минимальный интервал времени (в миллисекундах), в течение которого должна удерживаться нажатой

кнопка мыши, чтобы была инициирована операция буксировки. Значение по умолчанию – 0 миллисекунд.

9.2. Определение виджета

Как и в случае с виджетом MaxLength, созданным нами в предыдущей главе, необходимо выполнить некоторые подготовительные операции, прежде чем приступать к реализации самого виджета. Сначала определим будущую функциональность расширения и последовательность операций его создания. А затем объявим виджет и подключим поддержку мыши, предоставляемую библиотекой jQuery UI.

Все эти шаги рассматриваются в следующих подразделах.

9.2.1. Функциональность виджета Signature

Виджет Signature, который мы создадим в этой главе, позволяет выполнять подпись (signature), отслеживая перемещение указателя мыши над определенной областью веб-страницы (элемент `div` или `span`). Для отображения подписи в графическом виде он использует встроенный элемент `canvas` и может генерировать текстовое представление подписи для сохранения и последующего использования.

Элемент canvas

Язык JavaScript не предназначался для динамического управления графикой, тем не менее у нас есть возможность создать временное изображение для использования в качестве подписи. Элемент `canvas`, определяемый стандартом HTML5, предоставляет стандартную поверхность для рисования в пределах веб-страницы, которую можно использовать для отображения динамически сгенерированного изображения. Он имеет свой прикладной интерфейс, позволяющий управлять им из сценариев на языке JavaScript.

К сожалению, этот элемент не поддерживается старыми версиями Internet Explorer, поэтому придется написать дополнительный программный код поддержки данной функциональности в таких браузерах. Сценарий `ExplorerCanvas` (http://excanvas.sourceforge.net/) добавляет необходимую поддержку элемента `canvas` в старые браузеры. Подключите его к своей странице, как показано ниже:

```
<!--[if IE]>
<script type="text/javascript" src="js/excanvas.js"></script>
<![endif]-->
```

В JavaScript не поддерживается возможность создания файлов изображений, поэтому виджет будет создавать текстовую версию подписи, которую легко можно передать на сервер и сохранить. Для хранения подписи в текстовом виде мы выбрали формат JSON, по-

тому что преобразование подписи в этот формат сопряжено с относительно низкими накладными расходами, и он легко обрабатывается интерпретаторами JavaScript.

Подпись состоит из нескольких линий, каждая из которых определяется последовательностью точек, связанных отрезками. Изображение подписи преобразуется в массив линий, каждая из которых является массивом точек, представленных координатами *x* и *y*. Например, подпись, изображенная на рис. 9.2, в формате JSON выглядит, как показано в листинге 9.1.

Листинг 9.1. Версия подписи в формате JSON

```
{"lines":[[[38,85],[38,83],[38,82],[39,80],[40,76],
[41,73],[42,69],[42,65],[43,61],[44,58],[44,54],
[44,51],[45,48],[46,44],[48,41],[48,40],[49,37],
[51,36],[52,35],[53,34],[54,33],[55,33],[56,33],
[57,34],[58,36],[59,39],[61,43],[63,47],[65,52],
[66,55],[66,59],[66,64],[67,67],[67,70],[67,72],
[67,73],[68,74],[69,74],[70,74]],
[[41,62],[42,62],[43,62],[45,60],[48,59],[54,56],
[60,54],[66,52],[71,51],[77,50],[80,50],[83,49]]]}
```

Рис. 9.2 ❖ Создание подписи и преобразование ее в формат JSON

Когда подпись в формате JSON окажется на сервере, ее можно преобразовать в изображение, пригодное для обработки на стороне сервера такими инструментами, как приложения на Java или .NET, или сохранить в исходном текстовом формате. Приложение на языке Java для преобразования подписи в графический формат можно найти на веб-сайте книги. Виджет также поддерживает возможность повторного отображения подписи, хранящейся в формате JSON.

Виджет также позволяет стереть подпись, если во время рисования была допущена ошибка, и вернуть ее обратно, если она была сохранена, например, для нужд проверки заполнения формы.

Виджет Signature использует имя `signature` и название пространства имен `kbw` (как и виджет MaxLength). Программный код виджета будет храниться в файле с именем `jquery.signature.js`, а таблицы

стилей CSS – в файле jquery.signature.css. Общая схема реализации расширения будет показана в следующем разделе.

9.2.2. Устройство расширения Signature

Расширение состоит из нескольких функций, переопределяющих или расширяющих возможности, предоставляемые инфраструктурой поддержки виджетов и модулем Mouse. Все они перечислены в листинге 9.2, а вслед за листингом описывается, когда вызывается каждая из них в течение жизненного цикла расширения.

Листинг 9.2. Функциональная структура расширения

```
var signatureOverrides = {
    // Параметры настройки виджета
    options: {...},
    // Выполняет инициализацию элемента
    _create: function() {...},
    // Перерисовывает виджет
    _refresh: function(init) {...},
    // Очищает подпись
    clear: function(init) {...},
    // Синхронизирует изменения и отправку событий
    _changed: function(event) {...},
    // Обрабатывает собственные параметры настройки
    _setOptions: function(options) {...},
    // Определяет момент начала рисования
    _mouseCapture: function(event) {...},
    // Начинает рисование новой линии
    _mouseStart: function(event) {...},
    // Продолжает рисование линии
    _mouseDrag: function(event) {...},
    // Завершает рисование линии
    _mouseStop: function(event) {...},
    // Преобразует линии в формат JSON
    toJSON: function() {...},
    // Рисует подпись, хранящуюся в формате JSON
    draw: function(sigJSON) {...},
    // Проверяет наличие подписи
    isEmpty: function() {...},
    // Удаляет расширение
    _destroy: function() {...}
};

$.widget('kbw.signature', $.ui.mouse, signatureOverrides);
```

Прежде всего следует отметить, что расширение Signature подключается к одному или более элементам div веб-страницы с применением обычной для jQuery парадигмы «выбирай и действуй». В этот

момент инфраструктура поддержки виджетов создаст экземпляр объекта для каждого выбранного элемента и сохранит в нем информацию о текущем состоянии расширения. Частью этого объекта является множество параметров настройки, управляющих внешним видом и поведением. В число этих параметров входят унаследованные значения по умолчанию (переменная `options`), а также любые другие значения, переданные в вызов функции инициализации. Затем будет вызвана функция `_create`, чтобы дать вам возможность выполнить специфические настройки.

Если позднее пользователь пожелает изменить настройки, он сможет воспользоваться методом `option`, обращение к которому отображается в вызов функции `_setOptions`, с целью дать вам возможность обработать новые значения параметров. В данном случае необходимо вызвать функцию `_refresh`, которая применит новые настройки и перерисует виджет.

При попытке начать операцию рисования указателем мыши на элементе виджета функция `_mouseCapture` отвергает ее, если виджет находится в неактивном состоянии. В противном случае вызывается функция `_mouseStart`, в которой можно инициализировать рисование новой линии в подписи. Затем, в процессе движения указателя мыши, последует множество вызовов функции `_mouseDrag`, и по окончании рисования (когда пользователь отпустит кнопку мыши) вызывается функция `_mouseStop`. В ходе слежения за движением мыши расширение будет сохранять координаты промежуточных точек во внутреннем массиве для последующей обработки. По окончании рисования линии вызывается функция `_changed`, чтобы сообщить пользователю о наличии изменений через вызов обработчика события.

Чтобы извлечь текстовое представление подписи в формате JSON, пользователь должен обратиться к методу `toJSON`. Полученное текстовое значение можно затем отправить на сервер или использовать в другом месте. Передав подпись в формате JSON методу `draw`, можно повторно вывести подпись на поверхности элемента виджета.

```
var signature = $('#mysignature').signature('toJSON');
...
$('#mysignature').signature('draw', signature);
```

> Функции, имена которых не начинаются с символа подчеркивания (_), могут вызываться непосредственно, через инфраструктуру поддержки виджетов, для чего достаточно указать имя функции при обращении к расширению, как показано в примере выше, где вызываются функции `toJSON` и `draw`. Функции, имена которых начинаются с символа подчеркивания, считаются скрытыми и недоступны непосредственно.

Пользователь может определить, была ли создана подпись, обращением к методу `isEmpty`, что может пригодиться для организации проверки заполнения формы, а обращением к методу `clear` – стереть прежде сохраненные линии.

Если пользователь пожелает удалить экземпляр виджета, он сможет сделать это, вызвав метод `destroy`, который отображается в вызов функции `_destroy`. В результате этого вызова будут отменены все изменения, произведенные на этапе инициализации.

Чтобы приступить к разработке расширения, его необходимо сначала объявить и унаследовать в нем класс jQuery UI Mouse.

9.2.3. Объявление виджета

Как и в случае с виджетом MaxLength, необходимо организовать *сокрытие тонкостей реализации с использованием областей видимости* и отказаться *от предположения, что имя $ будет ссылаться на jQuery*, объявив анонимную функцию и немедленно вызвав ее:

```
(function($) { // Скрывает из видимости внешнего кода,
               // устраняет конфликты с идентификатором $
    ... здесь располагается остальной код
})(jQuery);
```

Далее следует определить новый виджет, как показано в листинге 9.3, и позволить инфраструктуре поддержки виджетов интегрировать свою функциональность в виджет.

Листинг 9.3. Объявление виджета

```
// ❶ Определение нового виджета
var signatureOptions = {

    // Глобальные параметры настройки виджета
    // ❷ Значения по умолчанию
    options: {
        background: '#ffffff', // Цвет фона

        ... Другие значения параметров по умолчанию
    },

    _create: function() {...},

    // Остальные функции виджета
};

/* ❸ Объявление нового виджета
      Сохраняет и отображает подпись.
      Зависит от jquery.ui.widget, jquery.ui.mouse. */
$.widget('kbw.signature', $.ui.mouse, signatureOverrides);
```

Сначала определяется объект с атрибутами ❶, посредством которых будут переопределяться или расширяться встроенные функциональные возможности инфраструктуры, включая параметры настройки ❷ со значениями по умолчанию и собственные методы. Затем выполняется объявление виджета вызовом функции `$.widget` ❸, которой передается имя виджета и его пространство имен, разделенные точкой (`.`). В последнем аргументе передается объект, переопределяющий или расширяющий функциональные возможности, наследуемые виджетом.

Напомню, что второй аргумент в этом вызове определяет, какие другие «классы» будут использоваться в качестве основы для данного виджета. В этом случае необходимо обеспечить возможность взаимодействий с мышью, поэтому мы передали во втором аргументе `$.ui.mouse`. Благодаря этому наш виджет унаследует все возможности виджета Mouse, а также возможности базового виджета Widget, который наследуется виджетом Mouse.

В соответствии с этим объявлением jQuery создаст новую функцию (`$.fn.signature`) и соединит ее с переопределениями, реализованными в вашем коде. А подключенные возможности виджета Mouse позволят данному виджету следить за любыми попытками рисовать линии с помощью мыши.

Однако, прежде чем виджет сможет откликаться на события от мыши, его необходимо подключить к какому-нибудь элементу страницы.

9.3. Применение расширения к элементу

Чтобы задействовать расширение, пользователь должен выбрать соответствующий элемент веб-страницы и применить к нему расширение. Инфраструктура поддержки виджетов в библиотеке jQuery UI выполнит все необходимые операции автоматически, прежде чем позволить вам приступить к настройке целевого элемента для конкретных нужд.

9.3.1. Инициализация, выполняемая инфраструктурой

Как и в случае с виджетом MaxLength, обсуждавшимся в предыдущей главе, когда выполняется применение расширения к элементу, инфраструктура поддержки виджетов сначала создает объект для хранения информации о состоянии расширения и подключает его к элементу

с помощью функции data в виде атрибута с именем, соответствующим имени расширения (kbw-signature – в версиях jQuery UI 1.9 и выше; signature – в версиях, предшествовавших jQuery UI 1.9).

Внутри этого объекта создаются два дополнительных атрибута: element и options. Первый из них – это ссылка на элемент, к которому подключено расширение, а второй является копией параметров настройки расширения, значения которых определяются значениями по умолчанию и любыми параметрами настройки, переданными в вызов функции инициализации. Кроме того, инфраструктура проверяет наличие расширения Metadata (https://github.com/jquery-orphans/jquery-metadata) в странице. Если это расширение было подключено, инфраструктура проверит с его помощью наличие дополнительных настроек в атрибутах выбранного элемента, опять же опираясь на имя расширения. По умолчанию атрибут class хранит следующие настройки:

```
<div id="sign" class="{signature: {guideline: true}}"></div>
```

> Автоматическая поддержка расширения Metadata была исключена в версии jQuery UI 1.10, но вы можете сами восстановить эту поддержку, как показано в разделе 8.9.

Помимо настроек, выполняемых инфраструктурой поддержки автоматически, часто бывает желательно выполнить собственные настройки при применении расширения к элементу.

9.3.2. Собственная инициализация

В процессе инициализации инфраструктура вызывает функцию _create, которую вы можете переопределить в своем расширении, как показано в листинге 9.4. Напомню, что внутри функции _create ссылка this ссылается на текущий экземпляр объекта виджета.

Листинг 9.4. Подключение и инициализация

```
/* Инициализирует новый экземпляр виджета signature. */
// ❶ Переопределение функции _create
_create: function() {
    // ❷ Добавить класс-метку
    this.element.addClass(this.widgetFullName || this.widgetBaseClass);
    // ❸ Инициализировать для работы в других браузерах
    try {
        this.canvas = $('<canvas width="' + this.element.width() +
            '" height="' + this.element.height() + '">' +
            this.options.notAvailable + '</canvas>')[0];
        this.element.append(this.canvas);
        //    Получить контекст рисования
```

```
            this.ctx = this.canvas.getContext('2d');
        }
        // ❹ Инициализировать для работы в IE
        catch (e) {
            $(this.canvas).remove();
            this.resize = true;
            this.canvas = document.createElement('canvas');
            this.canvas.setAttribute('width', this.element.width());
            this.canvas.setAttribute('height', this.element.height());
            this.canvas.innerHTML = this.options.notAvailable;
            this.element.append(this.canvas);
            // ❺ Инициализировать ExplorerCanvas
            if (G_vmlCanvasManager) { // Загрузить excanvas.js
                G_vmlCanvasManager.initElement(this.canvas);
            }
            // ❻ Получить контекст рисования
            this.ctx = this.canvas.getContext('2d');
        }
        // ❼ Перерисовать виджет
        this._refresh(true);
        // ❽ Инициализировать поддержку мыши
        this._mouseInit();
},
```

Здесь переопределяется функция `_create` ❶, которая вызывается инфраструктурой поддержки виджетов в ходе выполнения процедуры инициализации. Она добавляет класс-метку к текущему выбранному элементу, чтобы упростить его идентификацию в дальнейшем, используя для этого значение `this.widgetFullName` (`this.widgetBaseClass` в версиях jQuery UI ниже 1.9), устанавливаемое инфраструктурой ❷. Далее выполняются операции, специфические для данного расширения.

Если текущий браузер имеет встроенную поддержку элемента `canvas` ❸, его можно создать и добавить обычным для jQuery способом. Ссылка на вновь созданный элемент `canvas` сохраняется в текущем экземпляре объекта виджета присваиванием атрибуту `this.canvas`. Так как в процессе рисования подписи элемент `canvas` будет использоваться непрерывно, в атрибуте `this.ctx` сохраняется также ссылка на контекст рисования ❹.

Поскольку в разных версиях IE элемент `canvas` поддерживается по-разному, попытка создать его может возбудить исключение, которое необходимо обработать отдельно ❺. В IE сначала создается новый элемент `canvas`, затем его размеры устанавливаются так, чтобы они совпадали с размерами родительского элемента, то есть элемента, к которому применяется расширение Signature. В элемент `canvas`

необходимо добавить некоторое текстовое содержимое (извлекается из значения параметра) для отображения, если браузер не поддерживает `canvas`, например из-за того, что не удалось загрузить сценарий `ExplorerCanvas`. После добавления нового элемента в контейнер вызывается процедура инициализации из сценария `ExplorerCanvas` ❻. В результате этого последнего шага в элемент `canvas` будут добавлены функции, которые автоматически предоставляются более современными браузерами. Как и в других браузерах, в атрибуте `this.ctx` сохраняется также ссылка на контекст рисования ❹.

Во всех браузерах вызывается функция `_refresh` ❼, чтобы применить установленные настройки. Более подробно об этой функции рассказывается в разделе 9.4.3.

В заключение выполняется инициализация механизма взаимодействий с мышью, реализуемого модулем jQuery UI Mouse ❽. На этом этапе устанавливаются обработчики-обертки низкоуровневых событий от мыши, которые преобразуют последовательности этих событий в высокоуровневые операции буксировки, упрощая тем самым реализацию виджета.

Теперь расширение подключено к элементу страницы. Следующий шаг – реализовать возможность настройки расширения посредством изменения его параметров.

9.4. Параметры настройки

Часто пользователи надеются получить возможность определить внешний вид и поведение расширения за счет передачи параметров настройки в вызов функции инициализации. Попробуйте предугадать, что могут пожелать изменить пользователи, и *предусмотреть соответствующие параметры*. А позаботившись о присваивании *осмысленных значений по умолчанию*, вы позволите использовать свой виджет с минимумом настроек.

Для достижения этих целей необходимо:

- определить значения по умолчанию для всех параметров настройки;
- предусмотреть возможность чтения и изменения значений параметров;
- немедленно применять любые изменения;
- реализовать поддержку активного и неактивного состояний расширения.

Рассмотрим каждый из этих пунктов по очереди.

9.4.1. Значения настроек по умолчанию

Инфраструктура поддержки виджетов автоматически устанавливает начальные значения параметров настройки в экземпляре виджета, копируя все значения по умолчанию, указанные в объявлении виджета, и любые значения, передаваемые в вызов функции инициализации. В листинге 9.5 показано, как определяются значения по умолчанию для параметров настройки.

Листинг 9.5. Значения по умолчанию параметров настройки

```
// ❶ Глобальные значения по умолчанию параметров для signature
options: {
    background: '#ffffff', // Цвет фона
    color: '#000000', // Цвет линий подписи
    thickness: 2, // Толщина линий
    guideline: false, // Добавлять опорную линию для подписи или нет?
    guidelineColor: '#a0a0a0', // Цвет опорной линии
    guidelineOffset: 50, // Смещение опорной линии снизу
    guidelineIndent: 10, // Отступы опорной линии от боковых краев
    notAvailable: 'Your browser doesn\'t support signing',
        // Сообщение об ошибке, если элемент canvas не поддерживается
    syncField: null, // Селектор выбора текстового поля
                     // для вывода подписи в текстовом виде
    change: null // Обработчик события изменения подписи
},
```

Значения по умолчанию определяются в атрибуте прототипа виджета ❶. Чтобы упростить доступ к ним, вы можете отобразить этот атрибут непосредственно в множество значений по умолчанию.

Объявляя атрибут `options` ❶ в определении виджета, необходимо перечислить в нем все возможные параметры настройки и присвоить им значения по умолчанию. Это поможет также поместить описание всех параметров в одном месте. Результат изменения некоторых из этих параметров можно видеть на рис. 9.3.

Значения по умолчанию сохраняются инфраструктурой поддержки виджетов в атрибуте прототипа виджета. Чтобы упростить доступ к ним, вы можете отобразить этот атрибут непосредственно в множество значений по умолчанию:

```
$.kbw.signature.options = $.kbw.signature.prototype.options;
```

После этого можно будет переопределять значения по умолчанию для всех экземпляров расширения, как показано ниже, до применения расширения к каким-либо элементам:

```
$.extend($.kbw.signature.options,
    {background: '#FFFFE0', guideline: true});
```

Рис. 9.3 ❖ Параметры настройки виджета Signature: цвет линий и фона, толщина линий, добавлена опорная линия

Несмотря на то что инфраструктура поддержки виджетов также автоматически обрабатывает попытки доступа к значениям параметров, вам часто будет требоваться выполнять дополнительные операции при их изменении, чтобы обеспечить соответствие внешнего вида и поведения установленным настройкам.

9.4.2. Установка параметров

Инфраструктура поддержки виджетов автоматически обрабатывает попытки извлечения и изменения значений параметров. Как мы уже видели на примере виджета MaxLength, у нас есть возможность внедриться в процесс изменения значений параметров и предусмотреть выполнение своих операций. В виджете Signature не требуется предпринимать каких-либо специфических действий в ответ на изменение отдельных параметров, поэтому в нем функция `_setOption` не переопределяется. Но нам требуется выполнить обновление виджета после изменения любого параметра, чтобы привести внешний вид и

поведение в соответствие с установленными настройками, как показано в листинге 9.6.

Листинг 9.6. Обработка события изменения параметров

```
/* Обработка события изменения параметров.
   @param  options  (object) новые значения параметров */
// ❶ Переопределить функцию _setOptions
_setOptions: function(options) {
    // ❷ Вызвать базовый метод обработки этого события
    this._superApply(arguments);
    // ❸ Перерисовать виджет
    this._refresh();
},
```

Функция `_setOptions` переопределяется ❶ с целью обновить виджет при изменении любого из параметров настройки. Вы всегда должны вызывать унаследованную реализацию обработки события изменения параметров ❷, чтобы обеспечить сохранение новых значений параметров в экземпляре виджета. Затем следует вызвать собственную функцию `_refresh`, чтобы применить изменения к элементу, управляемому виджетом ❸. Эта функция описывается в следующем разделе.

Версии jQuery UI ниже 1.9

В версии jQuery UI 1.9 изменился способ вызова унаследованных функций. В новых версиях jQuery UI такие функции должны вызываться через ссылку на родительский класс, как показано ниже:

```
this._superApply(arguments);
```

В старых версиях вызов унаследованных функций должен производиться через прототип виджета, например так:

```
$.Widget.prototype._setOptions.apply(this, arguments);
```

Чтобы виджет мог работать и со старыми, и с новыми версиями jQuery UI, можно проверить наличие новой функции и использовать соответствующий способ вызова, например:

```
// Вызвать базовый метод виджета
if (this._superApply) {
    this._superApply(arguments);
}
else {
    $.Widget.prototype._setOptions.apply(this, arguments);
}
```

9.4.3. Реализация параметров настройки Signature

По мере изменения параметров настройки, которые в расширении Signature влияют на внешний вид виджета, их значения необходимо

заново применять к целевым элементам. Эту задачу решает собственная функция _refresh. Она вызывается при инициализации (см. раздел 9.3.1) и при изменении значений параметров (см. раздел 9.4.2). В листинге 9.7 показано, как она действует.

Листинг 9.7. Обновление внешнего вида и поведения виджета

```
/* Перерисовывает виджет signature.
   @param  init  (boolean, внутренний) true - если выполняется инициализация */
// ❶ Определение функции _refresh
_refresh: function(init) {
    // ❷ Обновить размеры canvas в IE
    if (this.resize) {
        var parent = $(this.canvas);
        $('div', this.canvas).css(
            {width: parent.width(), height: parent.height()});
    }
    // ❸ Применить новые настройки отображения
    this.ctx.fillStyle = this.options.background;
    this.ctx.strokeStyle = this.options.color;
    this.ctx.lineWidth = this.options.thickness;
    this.ctx.lineCap = 'round';
    this.ctx.lineJoin = 'round';
    // Очистить область рисования
    this.clear(init);
},
```

Функция _refresh ❶ переопределяет унаследованную версию. Так как имя функции начинается с символа подчеркивания (_), она не может вызываться непосредственно как метод.

Браузер IE не поддерживает элемент canvas в той мере, как хотелось бы, поэтому необходимо изменить его размеры так, чтобы они совпадали с размерами его контейнера ❷. Данная функция – самое подходящее место для этого, потому что она вызывается до того, как у пользователя появится возможность приступить к рисованию. Флаг this.resize устанавливается в функции _create (см. раздел 9.3.2) при выполнении инициализации виджета в IE.

Далее необходимо инициализировать контекст рисования последними значениями параметров настройки ❸, а затем очистить область рисования вызовом функции clear ❹ (см. раздел 9.7.1, где приводится описание этой функции). При последующем выводе элемента canvas будут использованы новые настройки.

Помимо изменения внешнего вида виджета, его можно также переключать в активное или неактивное состояние.

9.4.4. Активация и деактивация виджета

Активация (enabling) и деактивация (disabling) виджета являются стандартной функциональностью, предоставляемой инфраструктурой, и не требуют писать дополнительный код. Состояние виджета контролируется параметром `disabled`, который может принимать значения `true` (неактивное состояние) и `false` (активное состояние). Кроме того, инфраструктура поддерживает методы `enable` и `disable`, изменяющие значение этого параметра:

```
$('#sign').signature('disable');
...
$('#sign').signature('enable');
```

Внутри инфраструктура переключает два класса в главном элементе: `<пространство_имен>-<имя_расширения>-disabled` и `ui-state-disabled` – и устанавливает соответствующее значение в атрибуте `aria-disabled`. В данном виджете не требуется выполнять какие-либо дополнительные операции, непосредственно связанные с изменением его значения, но оно проверяется при попытке начать рисование линии (как показано в разделе 9.6.1).

Итак, мы реализовали возможность настройки расширения и применения изменений с целью привести внешний вид и поведение виджета в соответствие с ними. Некоторые из параметров настройки могут хранить ссылки на функции-обработчики, позволяя пользователям следить за наиболее важными событиями, происходящими в жизненном цикле виджета, как описывается в следующем разделе.

9.5. Добавление обработчиков событий

Виджет Signature дает пользователям возможность выполнять дополнительные операции в ответ на изменение рисунка с помощью события. Инфраструктура поддержки виджетов имеет возможность возбуждения событий посредством функции `_trigger`.

Чтобы подключить обработчик события к расширению, необходимо:

- дать пользователям возможность регистрировать свои обработчики событий;
- возбуждать события в соответствующие моменты времени.

В следующих подразделах описывается, как это реализуется на практике.

9.5.1. Регистрация обработчиков событий

Данным расширением поддерживается только одно событие: change. Оно возникает, когда завершается рисование новой линии, производится отображение подписи из текстового представления и при очистке области рисования.

Ссылка на обработчик change – это всего лишь один из параметров настройки расширения, как показано в листинге 9.8, по умолчанию имеющий значение null. Чтобы подключить свой обработчик события, пользователь должен присвоить этому параметру ссылку на свою функцию, принимающую два аргумента (как и другие обработчики событий в jQuery UI) – само событие и объект ui. В данном случае не требуется передавать обработчику какую-либо дополнительную информацию, поэтому значением аргумента ui всегда будет пустой объект. Внутри обработчика переменная this ссылается на главный элемент, к которому подключено расширение.

Листинг 9.8. Определение обработчика событий

```
// Настройки виджета signature по умолчанию
options: {
    ...
    change: null // Функция, которая вызывается при изменении
},
```

Пользователи могут зарегистрировать свой обработчик событий в процессе инициализации расширения, как показано в следующем фрагменте, или изменяя параметры настройки позднее:

```
$('#sign').signature({change: function(event, ui) {
    $('#submit').prop('disabled', false);
}});
```

При желании для подписки на события можно использовать стандартные функции bind и on из библиотеки jQuery, благодаря тому что инфраструктура поддержки виджетов способна автоматически отображать события в соответствующие им обработчики. Имя события определяется как комбинация из имени расширения и имени параметра (в данном случае signaturechange).

```
$('#sign').signature().on('signaturechange', function(event, ui) {
    $('#submit').prop('disabled', false);
}});
```

После установки обработчика событий необходимо обеспечить его вызов в соответствующие моменты времени.

9.5.2. Вызов обработчиков событий

Событие change возбуждается всякий раз, когда изменяется рисунок подписи: при очистке области рисования, по окончании рисования новой линии и при отображении подписи из текстового представления в формате JSON. В каждом из этих случае вызывается функция _changed, переопределяемая виджетом и представленная в листинге 9.9.

Листинг 9.9. Возбуждение события

```
/* Синхронизирует изменения и генерирует событие change.
   @param  event  (Event) генерируемое событие */
// ❶ Определение функции _changed
_changed: function(event) {
    // ❷ Синхронизировать изменения с содержимым текстового поля
    if (this.options.syncField) {
        $(this.options.syncField).val(this.toJSON());
    }
    // ❸ Возбудить событие
    this._trigger('change', event, {});
},
```

Здесь определяется функция _changed, которая будет вызываться при каждом изменении рисунка подписи ❶. Если в настройках определена ссылка на текстовое поле, которое должно содержать подпись в текстовом формате, содержимое этого поля необходимо привести в соответствие с последними изменениями ❷. Подробнее о функции toJSON рассказывается в разделе 9.7.2.

В заключение вызовом функции _trigger виджета генерируется событие ❸. В качестве аргументов ей передаются имя события, оригинальное событие, вызвавшее данное событие, и объект ui с информацией, определяемой самим виджетом. Событие, переданное функции _changed, и будет тем самым оригинальным событием от мыши, вызвавшим событие change. Никакой дополнительной информации не требуется передавать обработчику, поэтому в последнем аргументе передается пустой объект.

Покончив с инициализацией и настройкой расширения, можно приступать к реализации основной функциональности – сохранению информации о перемещениях указателя мыши.

9.6. Взаимодействие с мышью

Модуль jQuery UI Mouse автоматически перехватывает стандартные события от мыши и преобразует их высокоуровневые события буксировки. Он также учитывает настройки минимального смещения и

минимального интервала времени в данном виджете и разрешает следить за событиями буксировки только одному элементу.

Виджету остается только определить следующие обработчики событий буксировки:

- проверка возможности начать буксировку;
- начало операции буксировки;
- слежение за положением указателя в процессе буксировки;
- завершение операции буксировки.

Обсудим их по очереди.

9.6.1. Можно ли начать буксировку?

Модуль Mouse определяет момент, когда можно начать операцию буксировки, опираясь на целую комбинацию условий, включая: как далеко переместился указатель мыши и как долго удерживается кнопка мыши. Кроме того, модуль позволяет виджетам, использующим его, определять собственные условия за счет переопределения функции `_mouseCapture`, которая представлена в листинге 9.10. Эта функция принимает один аргумент – событие от мыши – и должна вернуть `true`, если операция буксировки может быть начата, и `false` в противном случае. Реализация по умолчанию всегда возвращает `true`.

Листинг 9.10. Проверка возможности начать буксировку

```
/* Проверка возможности начать буксировку.
   @param   event     (Event)  оригинальное событие от мыши
   @return  (boolean)  true - если возможно, false - если нет */
// ❶ Переопределение функции _mouseCapture
_mouseCapture: function(event) {
    // ❷ Запретить буксировку, если элемент неактивен
    return !this.options.disabled;
},
```

Чтобы определить собственное условие начала буксировки, мы переопределили функцию `_mouseCapture` ❶. Виджет Signature может начать слежение за буксировкой, только если он находится в активном состоянии. Но параметр настройки `disabled` определяет состояние «неактивности», поэтому функция возвращает инвертированное значение этого параметра ❷.

> В данной функции обработки события буксировки, как и во всех остальных функциях, рассматриваемых в этой главе, переменная `this` ссылается на текущий экземпляр объекта виджета, который среди прочих атрибутов хранит ссылку на целевой элемент (`this.element`) и коллекцию текущих параметров настройки (`this.options`).

9.6.2. Начало буксировки

Чтобы получить возможность выполнять собственные операции в момент начала буксировки, необходимо переопределить функцию `_mouseStart`, как показано в листинге 9.11. В аргументе `event` этой функции передается информация о местоположении указателя мыши в момент начала буксировки.

```
/* Начинает рисование новой линии.
   @param  event  (Event) оригинальное событие от мыши */
// ❶ Переопределить функцию _mouseStart
_mouseStart: function(event) {
    // ❷ Вычислить текущую позицию указателя мыши
    this.offset = this.element.offset();
    this.offset.left -= document.documentElement.scrollLeft ||
        document.body.scrollLeft;
    this.offset.top -= document.documentElement.scrollTop ||
        document.body.scrollTop;
    // ❸ Создать текущую точку
    this.lastPoint = [this._round(event.clientX - this.offset.left),
        this._round(event.clientY - this.offset.top)];
    // ❹ Создать новую линию и сохранить ее
    this.curLine = [this.lastPoint];
    this.lines.push(this.curLine);
},
```

Реализация по умолчанию функции `_mouseStart` ничего не делает, поэтому ее необходимо переопределить, чтобы обработать момент начала буксировки ❶. Координаты указателя мыши, что передаются этой функции в объекте события, относятся к системе координат окна браузера – видимой части страницы, поэтому их необходимо преобразовать в систему координат элемента `canvas`.

Вычисления начинаются с определения смещений целевого элемента по осям координат в видимой области ❷, с учетом того, что веб-страница, возможно, была прокручена. Смещения сохраняются в экземпляре объекта виджета для дальнейшего использования (`this.offset`). Опираясь на полученные смещения и координаты события от мыши, создается объект, представляющий точку в системе координат с началом в левом верхнем углу оригинального элемента ❸. Затем создается новая линия, содержащая лишь одну точку, потому что операция буксировки только началась, и добавляется в список линий ❹.

После создания новой линии можно начинать следить за движением указателя мыши и сохранять его координаты.

9.6.3. Слежение за положением указателя в процессе буксировки

Следить за движением указателя мыши в процессе буксировки можно, переопределив функцию _mouseDrag. Она также получает аргумент event с информацией о местоположении указателя мыши. В листинге 9.12 демонстрируется реализация этой функции в виджете Signature.

Листинг 9.12. Слежение за движением указателя мыши

```
/* Следит за указателем мыши.
   @param  event  (Event) оригинальное событие от мыши */
// ❶ Переопределение функции _mouseDrag
_mouseDrag: function(event) {
    // ❷ Вычислить текущее положение указателя мыши
    var point = [this._round(event.clientX - this.offset.left),
        this._round(event.clientY - this.offset.top)];
    this.curLine.push(point);
    // ❸ Продолжить рисование от предыдущей позиции
    this.ctx.beginPath();
    this.ctx.moveTo(this.lastPoint[0], this.lastPoint[1]);
    this.ctx.lineTo(point[0], point[1]);
    this.ctx.stroke();
    // ❹ Сохранить новую последнюю позицию
    this.lastPoint = point;
},
```

Чтобы иметь возможность выполнять свои операции в процессе движения указателя мыши, необходимо переопределить функцию _mouseDrag ❶. Так как координаты указателя мыши в объекте события относятся к системе координат страницы, их необходимо преобразовать в систему координат элемента canvas, опираясь на смещения, вычисленные в момент начала буксировки ❷, а полученные координаты добавить в виде объекта точки в массив, определяющий текущую линию.

Смещение указателя мыши необходимо отобразить в элементе canvas ❸. Используя контекст рисования, мы рисуем прямую линию, соединяющую предыдущую и текущую точки. Параметры оформления линии были установлены в процессе инициализации (или позднее, посредством изменения соответствующих параметров) и задействованы функцией _refresh (см. раздел 9.4.3). В заключение текущая точка сохраняется как последняя ❹.

Мониторинг перемещения указателя мыши позволяет сохранять линии, нарисованные пользователем. По окончании рисования линии также необходимо выполнить некоторые операции.

9.6.4. Завершение буксировки

По завершении операции буксировки вызывается функция `_mouseStop`. Чтобы выполнить свои операции, следует переопределить эту функцию. Информация о местоположении указателя мыши передается этой функции в аргументе `event`. В листинге 9.13 приводится полная реализация функции в виджете Signature.

Листинг 9.13. Завершение буксировки

```
/* Конец создания линии.
   @param  event  (Event) оригинальное событие от мыши */
// ❶ Переопределение функции _mouseStop
_mouseStop: function(event) {
    // ❷ Очистить последнюю точку и текущую линию
    this.lastPoint = null;
    this.curLine = null;
    // ❸ Вызвать функцию обработки изменений виджета
    this._changed(event);
},
```

Чтобы выполнить собственные операции по окончании операции буксировки, необходимо переопределить функцию `_mouseStop` ❶. В этой функции мы сначала стираем информацию о последней точке и текущей линии ❷, а затем вызываем функцию, выполняющую обработку события изменения виджета ❸, чтобы синхронизировать поле с текстовым представлением подписи и возбудить событие `change` (см. раздел 9.5.2).

Теперь наше расширение способно сохранять подпись, нарисованную пользователем на элементе с помощью мыши, в массиве линий. Но нужно предусмотреть еще кое-какие операции, чтобы обеспечить возможность использовать сохраненные точки в дальнейшем.

9.7. Добавление методов

Расширение Signature предоставляет дополнительные функциональные возможности для выполнения операций с сохраненной подписью. В их число входят: очистка подписи, преобразование подписи в текстовый формат, пригодный для передачи и хранения, вывод подписи, представленной в текстовом формате, и определение наличия хотя бы одной линии в подписи.

Напомню, что инфраструктура поддерживает вызов методов расширений, отображая их непосредственно в имена функций экземпляра объекта виджета. Любая функция, имя которой начинается

с символа подчеркивания (_), считается внутренней и не может быть вызвана подобным образом. Если метод возвращает некоторое значение, оно будет передано вызывающему коду. В противном случае инфраструктура вернет текущую коллекцию jQuery, чтобы дать возможность продолжить цепочку вызовов.

9.7.1. Очистка подписи

Если пользователь допустил ошибку при создании подписи или остался недоволен результатом, он может стереть подпись и начать все сначала. Эту возможность реализует метод `clear`.

Листинг 9.14. Очистка подписи

```
/* Очищает область рисования подписи.
   @param  init  (boolean, внутренний) true - если выполняется инициализация */
// ❶ Функция, реализующая метод clear
clear: function(init) {
    // ❷ Очистить область рисования
    this.ctx.fillRect(0, 0,
        this.element.width(), this.element.height());
    // ❸ Нарисовать опорную линию, если требуется
    if (this.options.guideline) {
        this.ctx.save();
        this.ctx.strokeStyle = this.options.guidelineColor;
        this.ctx.lineWidth = 1;
        this.ctx.beginPath();
        this.ctx.moveTo(this.options.guidelineIndent,
            this.element.height() - this.options.guidelineOffset);
        this.ctx.lineTo(
            this.element.width() - this.options.guidelineIndent,
            this.element.height() - this.options.guidelineOffset);
        this.ctx.stroke();
        this.ctx.restore();
    }
    // ❹ Очистить массив линий
    this.lines = [];
    // ❺ Вызвать событие изменения виджета, если это не инициализация
    if (!init) {
        this._changed();
    }
},
```

Имя функции совпадает с именем соответствующего ей метода ❶, благодаря чему обеспечивается автоматическое отображение обращения к методу в вызов функции инфраструктурой поддержки виджетов. Очистка области рисования выполняется за счет рисования закрашенного прямоугольника посредством сохраненного

контекста и с использованием параметра настройки цвета фона (см. листинг 9.7) ❷.

Если настроено рисование опорной линии, производится рисование этой линии указанного цвета и в указанной позиции ❸. Стиль оформления линий самой подписи сохраняется в текущем контексте рисования вызовом функции `save()`, затем рисуется опорная линия, и после этого восстанавливается стиль оформления линий подписи вызовом функции `restore()`.

Далее очищается массив линий ❹ и вызывается функция `_changed` ❺, чтобы синхронизировать текстовое представление с графическим изображением (если необходимо) и сообщить пользователю об изменениях. Обработка события `changed` не требуется на этапе инициализации виджета, поэтому она не производится при установленном флаге `init`. Хотя эта функция имеет аргумент, он передается только при внутреннем использовании функции – конечному пользователю не требуется определять его.

Вызывать это метод можно, как показано ниже:

```
$('#sign').signature('clear');
```

9.7.2. Преобразование в формат JSON

Массив линий, образующих подпись, может быть представлен в виде объекта JSON, чтобы упростить его передачу и хранение. Этот объект имеет единственный атрибут (`lines`) – массив линий подписи. Каждая линия представлена массивом точек (с координатами *x* и *y*). Для получения текстовой версии такого объекта JSON используется метод `toJSON`, реализация которого представлена в листинге 9.15.

Листинг 9.15. Преобразование в формат JSON

```
/* Преобразует массив линий в текстовое представление в формате JSON.
   @return  (string) текстовое представление линий в формате JSON */
// ❶ Функция, реализующая метод toJSON
toJSON: function() {
    // ❷ Добавить каждую линию
    return '{"lines":[' + $.map(this.lines, function(line) {
            // ❸ Добавить каждую точку
            return '[' + $.map(line, function(point) {
                return '[' + point + ']';
            }) + ']';
    }) + ']}';
},
```

Реализация метода `toJSON` оформлена в виде одноименной функции ❶. Процесс преобразования начинается с определения внешнего

объекта с единственным атрибутом lines, затем производится поочередное добавление в текст в формате JSON всех линий ❷. Для каждой линии последовательно добавляются составляющие их точки ❸.

> Это лишь частичная реализация преобразования JSON в строку. Полноценную версию можно найти на сайте GitHub: https://gist.github.com/JaNightmare/2051416 в виде функции stringifyJSON.

Получить JSON-версию подписи можно следующим вызовом:

```
var jsonText = $('#sign').signature('toJSON');
```

Полученная в результате строка может выглядеть, как показано ниже, где определяются две линии с множеством точек в каждой:

```
{"lines":[[[38,85],[38,83],[38,82],[39,80],[40,76],
[41,73],[42,69],[42,65],[43,61],[44,58],[44,54],
[44,51],[45,48],[46,44],[48,41],[48,40],[49,37],
[51,36],[52,35],[53,34],[54,33],[55,33],[56,33],
[57,34],[58,36],[59,39],[61,43],[63,47],[65,52],
[66,55],[66,59],[66,64],[67,67],[67,70],[67,72],
[67,73],[68,74],[69,74],[70,74]],
[[41,62],[42,62],[43,62],[45,60],[48,59],[54,56],
[60,54],[66,52],[71,51],[77,50],[80,50],[83,49]]]}
```

Как вариант можно установить параметр synchField виджета, идентифицирующий текстовое поле для вывода текстовой версии в формате JSON, синхронно с изменением изображения подписи. В этом параметре можно указать строку селектора jQuery, ссылку на элемент DOM или на существующий объект jQuery. Синхронизация производится в функции _changed (см. раздел 9.5.2).

Итак, поскольку теперь поддерживается возможность преобразования подписи в текстовое представление в формате JSON для последующего сохранения, значит, позднее может потребоваться выполнить обратное преобразование, чтобы повторно вывести сохраненную ранее подпись.

9.7.3. Повторное отображение подписи

Для повторного отображения подписи можно использовать текстовое представление в формате JSON. Метод draw принимает строку JSON или объект и отображает подпись, определяемую этой строкой или объектом в элементе canvas, как показано в листинге 9.16.

Листинг 9.16. Повторное отображение подписи

```
/* Рисует подпись, извлекая ее из строки в формате JSON.
    @param  sigJSON  (object) объект с атрибутом lines,
```

```
                                в котором хранится массив точек, или
                        (string) текстовая версия в формате JSON */
// ❶ Функция, реализующая метод draw
draw: function(sigJSON) {
    // ❷ Очистить поверхность рисования
    this.clear(true);
    // ❸ Скопировать определение линий
    if (typeof sigJSON == 'string') {
        sigJSON = $.parseJSON(sigJSON);
    }
    this.lines = sigJSON.lines || [];
    var ctx = this.ctx;
    // ❹ Нарисовать каждую линию
    $.each(this.lines, function() {
        ctx.beginPath();
        $.each(this, function(i) {
            ctx[i == 0 ? 'moveTo' : 'lineTo'](this[0], this[1]);
        });
        ctx.stroke();
    });
    // ❺ Возбудить событие chage
    this._changed();
},
```

Реализация метода `draw` оформлена в виде одноименной функции `draw` ❶. Она принимает единственный аргумент – объект с массивом линий, составляющих подпись, или строку с текстовым представлением подписи в формате JSON.

Сначала функция очищает область рисования вызовом функции `clear` ❷. Если подпись передается в виде строки, ее необходимо преобразовать в эквивалентный объект JavaScript ❸. Иначе используется исходный объект, полученный в виде аргумента. Затем определения линий из этого объекта переносятся в экземпляр объекта виджета и выводятся по точкам ❹. Напомню, что при рисовании используются стили оформления линий, установленные прежде в функции `_refresh` (см. раздел 9.4.3). После рисования требуется выполнить синхронизацию и послать пользователю событие `change` вызовом функции `_changed` ❺ (см. раздел 9.5.2).

9.7.4. Проверка наличия подписи

Может так получиться, что область рисования подписи окажется обязательным для заполнения полем ввода. Чтобы помочь упростить процедуру проверки заполнения поля и скрыть внутренние особенности реализации виджета, в нем имеется метод `isEmpty` (листинг 9.17), возвращающий логическое значение, как признак наличия подписи.

Листинг 9.17. Проверка наличия подписи

```
/* Определяет присутствие в элементе canvas хотя бы одной линии.
   @return  (boolean) true - если подпись отсутствует,
                      false - если присутствует */
// ❶ Функция, реализующая метод isEmpty
isEmpty: function() {
    // ❷ Вернуть признак пустого поля
    return this.lines.length == 0;
},
```

Метод `isEmpty` реализован в виде одноименной функции ❶. Она возвращает признак пустой подписи, сравнивая количество имеющихся линий с нулем ❷.

Обратите внимание, что этот метод возвращает значение типа `Boolean` и не может использоваться в середине цепочки вызовов функций jQuery. Ниже показан пример использования метода `isEmpty`, а на рис. 9.4 – результат работы этого примера:

```
if ($('#sign').signature('isEmpty')) {
    alert('Please enter your signature');
}
```

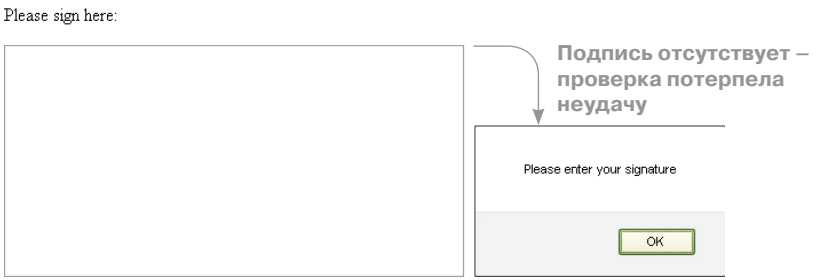


Рис. 9.4 ❖ Результат проверки наличия подписи

На этом завершается реализация методов взаимодействий с расширением Signature. Но вам еще может потребоваться удалить виджет Signature.

9.8. Удаление виджета

Чтобы удалить все следы пребывания виджета, пользователи вызывают метод `destroy`. Как и в случае с другими методами, обращение к нему отображается в вызов функции с тем же именем, объявленной внутри расширения.

9.8.1. Метод _destroy

Инфраструктура поддержки виджетов содержит функцию destroy, которая устраняет все изменения, произведенные на этапе инициализации. Она автоматически вызывает функцию _destroy в нужные моменты времени, реализация по умолчанию которой ничего не делает. Чтобы реализовать собственный метод destroy, необходимо переопределить унаследованную функцию _destroy и внутри нее отменить все изменения, выполненные функцией _create, как показано в листинге 9.18.

Листинг 9.18. Удаление виджета

```
/* Удаляет функциональные возможности расширения signature. */
// ❶ Определение функции, реализующей метод destroy
_destroy: function() {
    // ❷ Удалить класс-метку
    this.element.removeClass(
        this.widgetFullName || this.widgetBaseClass);
    // ❸ Удалить элемент canvas
    $(this.canvas).remove();
    this.canvas = this.ctx = this.lines = null;
    // ❹ Удалить поддержку мыши
    this._mouseDestroy();
}
```

Здесь объявляется функция _destroy, переопределяющая встроенный метод destroy ❶. Внутри функции нам необходимо удалить класс-метку ❷, элемент canvas и связанные с ним ссылки ❸, добавленные на этапе настройки. Вызовом функции _mouseDestroy также удаляется поддержка мыши ❹.

После вызова этого метода из выбранных элементов удаляются функциональные возможности расширения Signature, и они возвращаются в исходное состояние.

Версии jQuery UI ниже 1.9

В версиях jQuery UI ниже 1.9 вместо функции _destroy необходимо переопределять функцию destroy и вызывать унаследованную функцию через прототип виджета. Чтобы виджет мог работать и со старыми, и с новыми версиями jQuery UI, в старых версиях необходимо переопределить функцию destroy и заставить ее вызывать функцию _destroy, как это делается в последних версиях. Может также потребоваться вызывать унаследованную функцию destroy, реализованную в инфраструктуре поддержки виджетов. Как это делается, показано ниже:

```
if (!$.Widget.prototype._destroy) {
    $.extend(maxlengthOverrides, {
```

```
        /* Удалить функциональные возможности
           расширения maxlength из элемента. */
        destroy: function() {
            this._destroy();
            // Вызвать базовый метод виджета
            $.Widget.prototype.destroy.call(this);
        }
    });
}
```

9.9. Законченное расширение

Итак, мы закончили разработку виджета Signature для jQuery UI, позволяющего рисовать подпись в специальной области веб-страницы, преобразовывать ее в текстовый вид для дальнейшей обработки и восстанавливать из текстового представления. Взаимодействия с мышью существенно упрощаются благодаря применению модуля jQuery UI Mouse. Полный код расширения доступен для загрузки на сайте книги.

Это нужно знать

Наследуйте модуль jQuery UI Mouse, когда возникнет необходимость использовать в виджете операции буксировки мышью.

Вызывайте `_mouseInit` для инициализации механизма взаимодействий с мышью и `_mouseDestroy` – для его удаления.

Переопределяйте функцию `_mouseCapture`, чтобы получить возможность проверить соблюдение необходимых условий перед началом операции буксировки.

Переопределяйте функции `_mouseStart`, `_mouseDrag` и `_mouseStop`, чтобы получить возможность реагировать на события, возникающие в ходе буксировки.

Для создания, настройки и уничтожения расширения все еще можно использовать стандартные функции и методы инфраструктуры поддержки виджетов.

Попробуйте сами

Создайте расширение, имитирующее простой графический редактор, также основанное на применении элемента `canvas`. Реализуйте возможность рисования прямоугольника и его перемещения в области рисования с помощью мыши. Для сохранения изображения в процессе буксировки можете использовать элемент `Image`, чтобы упростить вывод прямоугольника:

```
this.img.src = this.canvas.toDataURL(); // Сохранить начальное состояние
this.ctx.drawImage(this.img, 0, 0); // Восстановить начальное состояние
```

9.10. В заключение

Библиотека jQuery включает поддержку взаимодействий с мышью посредством низкоуровневых событий, однако библиотека jQuery UI предоставляет более удобный высокоуровневый интерфейс для выполнения операций буксировки мышью. Модуль jQuery UI Mouse перехватывает стандартные события `mousedown`, `mousemove` и `mouseup` и преобразует их в вызовы функций `_mouseStart`, `_mouseDrag` и `_mouseStop`, позволяя сосредоточиться на реализации реакции в ответ на перемещения указателя мыши, не задумываясь об особенностях работы механизма, вызывающего их. Операция буксировки мышью используется в нескольких виджетах, входящих в состав библиотеки jQuery UI.

Для начала необходимо добавить модуль Mouse в объявление виджета. Затем переопределить перечисленные выше функции, вызываемые в процессе буксировки. В реализации расширения можно использовать все функциональные возможности модуля Widget, потому что он наследуется модулем Mouse.

На примере создания расширения Signature в этой главе демонстрируется, как использовать механизм взаимодействий с мышью для внедрения новых функциональных возможностей в веб-страницы. Это расширение дает возможность создать подпись в специальной области страницы, рисуя ее с помощью мыши, и затем преобразовать эту подпись в текстовое представление в формате JSON, пригодное для хранения и дальнейшей обработки.

В последней главе этой части книги мы познакомимся с инфраструктурой эффектов jQuery UI и посмотрим, как с ее помощью воспроизводить различные анимационные эффекты на веб-страницах.

Глава 10

Эффекты jQuery UI

Эта глава охватывает следующие темы:

- инфраструктура поддержки эффектов в библиотеке jQuery UI;
- добавление нового эффекта;
- что такое «функции управления переходами»;
- добавление новых функций управления переходами.

Наряду с различными виджетами (упоминавшимися в двух предыдущих главах) в составе библиотеки jQuery UI имеются дополнительные функции, реализующие различные механизмы взаимодействий с пользователем, такие как `draggable` и `droppable`, а также визуальные эффекты для применения к элементам. Эти эффекты дополняют процесс отображения и сокрытия элементов веб-страницы или служат для привлечения внимания к конкретным элементам, изменяя различные аспекты их внешнего вида с течением времени. Встроенные эффекты основываются на базовой функциональности библиотеки. Как вы уже, наверное, догадались, у вас есть возможность создавать собственные анимационные эффекты и интегрировать их в библиотеку jQuery UI.

Аналогично *функции управления переходами* (easings) расширяют анимационные эффекты, определяя, как будет изменяться значение атрибута с течением времени. Помимо двух стандартных функций управления переходами в jQuery, в библиотеке jQuery UI имеется еще множество различных подобных функций. Вы также можете добавлять собственные функции переходов, чтобы реализовать поведение анимационных эффектов, в точности соответствующее вашим пожеланиям.

Все вместе эти визуальные эффекты могут оживить страницу и придать неповторимый внешний вид вашему веб-сайту. Вы можете заставить элемент сворачиваться в точку при его сокрытии, подобно тому, как исчезает изображение на экране телевизора при его выключении. Или привлечь внимание пользователя к элементу, содержимое которого обновилось в результате получения новых данных от сервера, изменив на короткое время цвет фона. Для имитации движения

физических тел можно использовать функции управления переходами при перемещении элементов. Например, можно сымитировать подпрыгивание элемента так, как если бы на него действовала сила тяжести. Эффекты обеспечивают привлечение внимания пользователя и порождают у него более благоприятные впечатления от работы с веб-страницей.

10.1. Инфраструктура поддержки эффектов в jQuery UI

Инфраструктура поддержки эффектов в библиотеке jQuery UI, как и инфраструктура поддержки виджетов, имеет модульную архитектуру, позволяя выбирать нужные компоненты и уменьшить сетевой трафик. Имеется даже возможность скомпоновать собственную версию библиотеки для загрузки с учетом зависимостей между модулями (http://jqueryui.com/download).

Прежде чем приступить к созданию новых эффектов, необходимо познакомиться с возможностями, предлагаемыми инфраструктурой поддержки эффектов, чтобы потом использовать их в собственных эффектах.

10.1.1. Модуль Effects

В основе всех модулей эффектов, имеющихся в библиотеке jQuery UI, лежит модуль Effects. Он содержит все необходимые функциональные возможности, поэтому у вас не возникнет необходимости повторно изобретать колесо, и вы сможете использовать их непосредственно в своих эффектах. Помимо эффекта анимации цвета, вы найдете в этом модуле средства анимации значений атрибутов от одного класса к другому, а также несколько низкоуровневых функций, которые могут пригодиться при создании новых эффектов.

Анимация цвета

Модуль Effects добавляет анимационный эффект, способный изменять атрибуты стиля, содержащие значение цвета: цвета переднего плана, фона и рамки. Сама библиотека jQuery поддерживает анимационные эффекты, основанные только на изменении атрибутов с простыми числовыми значениями, возможно, содержащими необязательные обозначения единиц измерения, такие как `px`, `em` или `%`. Она не поддерживает более сложные значения, такие как, например, цвет,

и не знает, как правильно изменять такие значения для достижения желаемого эффекта, как, например, переход от синего цвета к красному через фиолетовый.

Значения цвета состоят из трех компонентов: красной, зеленой и синей составляющих, каждая из которых выражается числом в диапазоне от 0 до 255. В коде HTML и CSS значения цвета могут определяться в разных форматах:

- шестнадцатеричными цифрами: `#DDFFE8`;
- малыми шестнадцатеричными цифрами: `#CFC`;
- десятичными значениями в формате RGB: `rgb(221, 255, 232)`;
- десятичными процентами в формате RGB: `rgb(87%, 100%, 91%)`;
- десятичными значениями в формате RGB с указанием прозрачности: `rgba(221, 255, 232, 127)`;
- названиями: `lime`.

При воспроизведении анимационного эффекта красная, зеленая и синяя составляющие должны изменяться отдельно друг от друга, от своих начальных значений к конечным, и объединяться в общее значение цвета на каждом промежуточном шаге.

Библиотека jQuery UI добавляет дополнительные этапы для каждого изменяемого атрибута с целью обеспечить правильное декодирование текущего и конечного цветов и изменения значений атрибутов. В дополнение к форматам представления цвета, перечисленным выше, функция `animate` может также принимать массив с тремя числовыми значениями (каждое в диапазоне от 0 до 255), определяющими цвет. Благодаря этой функции вы можете плавно изменять цвет, как простое числовое значение:

```
$('#myDiv').animate({backgroundColor: '#DDFFE8'});
```

Библиотека jQuery UI поддерживает расширенный список названий цветов, от простых `red` (красный) и `green` (зеленый) до более замысловатых `darkorchid` (темно-орхидейный) и darksalmon (темно-оранжево-розовый). Существует даже цвет `transparent` (прозрачный).

В главе 11 вы узнаете, как можно добавлять функции анимации для других нечисловых атрибутов.

Анимация класса

Стандартная библиотека jQuery позволяет добавлять, удалять или переключать классы в выбранных элементах. А библиотека jQuery UI пошла еще дальше – она позволяет воспроизводить эффекты перехода от одного состояния к другому.

Достигается это путем извлечения значений всех атрибутов, пригодных для анимации (числа и цвета) из начальной и конечной конфигураций и вызовом стандартной функции `animate` со всеми этими свойствами. Воспроизведение такого анимационного эффекта запускается передачей значения продолжительности эффекта стандартным функциям `addClass`, `removeClass` и `toggleClass`:

```
$('#myDiv').addClass('highlight', 1000);
```

Кроме того, jQuery UI добавляет новую функцию, `switchClass`, которая удаляет один класс и добавляет другой, выполняя плавный переход между двумя состояниями (если указана продолжительность перехода):

```
$('#myDiv').switchClass('oldClass', 'newClass', 1000);
```

10.1.2. Общие функции эффектов

Для поддержки различных эффектов, имеющихся в библиотеке jQuery UI, модуль Effects содержит несколько общих функций, которые, возможно, пригодятся и вам для реализации ваших эффектов. Для иллюстрации применения некоторых из этих функций в листинге 10.1 демонстрируются фрагменты реализации эффекта `slide` (сдвиг).

Листинг 10.1. Применение общих функций в реализации эффекта slide

```
$.effects.effect.slide = function( o, done ) {

    // Создать элемент
    var el = $( this ),
        props = [ "position", "top", "bottom",
            "left", "right", "width", "height" ],
        // ❶ Определить режим работы
        mode = $.effects.setMode( el, o.mode || "show" ),
        ...;

    // Подготовиться
    // ❷ Сохранить текущие значения
    $.effects.save( el, props );
    el.show();
    distance = o.distance || el[ ref === "top" ?
        "outerHeight" : "outerWidth" ]( true );

    // ❸ Создать контейнер для анимационного эффекта
    $.effects.createWrapper( el ).css({overflow: "hidden"});
    ...

    // Анимация
```

```
    animation[ ref ] = ...;

    // Воспроизвести
    el.animate( animation, {
        queue: false,
        duration: o.duration,
        easing: o.easing,
        complete: function() {
            if ( mode === "hide" ) {
                el.hide();
            }
            // ❹ Восстановить первоначальные значения
            $.effects.restore( el, props );
            // ❺ Удалить контейнер
            $.effects.removeWrapper( el );
            done();
        }
    });
};
```

Функция `setMode` ❶ используется для преобразования режима `toggle` в атрибуте `mode` в соответствующее значение `show` или `hide`, исходя из текущего состояния видимости элемента (`el`, объект jQuery). Если в атрибуте указано значение `show` или `hide`, сохраняется это значение или, если такой атрибут вообще отсутствует, передается значение по умолчанию `show`.

Перед запуском анимационного эффекта бывает нелишне вызвать функцию `save` ❷, чтобы сохранить значения некоторых атрибутов (имена которых перечислены в массиве `props`) элемента (`el`) для их восстановления по завершении. Значения будут сохранены в указанном элементе с использованием функции `data`.

Для упрощения реализации эффекта целевой элемент можно заключить в контейнер вызовом функции `createWrapper` ❸ и использовать его как начало координат для перемещения. Информация о местоположении копируется из указанного элемента (`el`) в элемент-обертку, чтобы он оказался непосредственно поверх оригинального элемента. Затем элемент позиционируется внутри нового контейнера по его левому верхнему углу, чтобы общий эффект был незаметен для пользователя. Функция возвращает ссылку на контейнер. Любые изменения в атрибутах `left/right/top/bottom` оригинального элемента теперь будут интерпретироваться относительно его оригинальной позиции, без оказания влияния на окружающие элементы.

Если значения атрибутов были предварительно сохранены, по окончании воспроизведения эффекта можно воспользоваться функ-

цией restore, чтобы восстановить в них исходные значения ❹. В этот же момент следует удалить контейнер вызовом функции removeWrapper ❺. Эта функция возвращает ссылку на контейнер, если он был удален, или сам элемент, если он не был заключен в контейнер.

В модуле jQuery UI Effects существуют также и другие функции, которые могут вам пригодиться.

- getBaseline(origin, original) – нормализует координаты origin (массив из двух элементов) в дробные значения (от 0.0 до 1.0) с учетом размеров original (объекта с атрибутами height и width). Преобразует именованные позиции (top, left, center и прочие) в значения 0.0, 0.5 или 1.0, а числовые – в пропорции, соответствующие размерам объекта. Возвращает объект с атрибутами x и y, в которых хранятся дробные значения для соответствующих координат. Например:

```
getBaseline(['middle', 20],
    {height: 100, width: 200}); // baseline = {x: 0.1, y: 0.5}
```

- setTransition(element, list, factor, value) – используется для масштабирования сразу нескольких атрибутов. Эта функция извлекает текущее значение каждого атрибута элемента element из перечисленных в списке list и умножает на factor. Результат сохраняется в объекте value в атрибутах с теми же именами. По окончании умножения всех атрибутов объект value возвращается вызывающему коду. Например, уменьшить некоторые значения наполовину можно, как показано ниже:

```
el.from = $.effects.setTransition(el,
    ['borderTopWidth', 'borderBottomWidth', ...], 0.5, el.from);
```

- cssUnit(key) – разбивает значение указанного атрибута (key) на числовую величину и единицы измерения (em, pt, px или %). Возвращает массив с двумя элементами. Если в атрибуте указаны неизвестные единицы измерения, возвращается пустой массив. Например:

```
var value = el.cssUnit('width'); // вернет, например, value =
[200, 'px']
```

Функции, представленные в этом разделе, используются многими эффектами в библиотеке jQuery UI. Мы познакомимся с подобными эффектами в следующем разделе, а потом посмотрим, как создать собственный эффект.

10.1.3. Существующие эффекты

В библиотеке jQuery UI реализовано множество эффектов. Большинство их них предназначено для управления процессом появления или сокрытия элементов (такие как `blind` и `drop`), тогда как другие служат для привлечения внимания к элементу (такие как `highlight` и `shake`). В табл. 10.1 перечислены все имеющиеся эффекты, а также параметры, которые можно использовать для изменения их поведения.

Таблица 10.1. Эффекты jQuery UI

Имя	Производимый эффект	Параметры
`blind`	Элемент появляется/исчезает, разворачиваясь/сворачиваясь вертикально (по умолчанию) или горизонтально, сверху или слева	`direction`
`bounce`	Элемент появляется/исчезает с имитацией подпрыгивания несколько раз	`direction, distance, times`
`clip`	Элемент появляется/исчезает, расширяясь/сокращаясь от центральной линии по вертикали (по умолчанию) или по горизонтали	`direction`
`drop`	Элемент появляется/исчезает, въезжая/выезжая из области видимости слева/влево (по умолчанию) или сверху/вверх, одновременно проявляясь от полной прозрачности или растворяясь.до полной прозрачности	`direction, distance`
`explode`	Элемент появляется/исчезает, собираясь/взрываясь из/на нескольких фрагментов	`pieces`
`fade`	Элемент появляется/исчезает, проявляясь от полной прозрачности или растворяясь до полной прозрачности	-
`fold`	Элемент появляется/исчезает, разворачиваясь/сворачиваясь сначала в одном направлении, затем в другом (по умолчанию: сначала по вертикали, затем по горизонтали)	`horizFirst, size`
`highlight`	Элемент изменяет цвет фона на короткое время	`color`
`puff`	Элемент появляется/исчезает, уменьшаясь/увеличивась в размерах, проявляясь от полной прозрачности или растворяясь до полной прозрачности	`direction, from, origin, percent, restore, to`
`pulsate`	Элемент растворяется и проявляется несколько раз	`times`
`scale`	Элемент увеличивается/уменьшается пропорционально из/в центральной точке на указанное число процентов	`direction, from, origin, percent, restore, scale, to`
`shake`	Элемент смещается из стороны в сторону несколько раз, имитируя встряхивание	`direction, distance, times`

Таблица 10.1 (окончание)

Имя	Производимый эффект	Параметры
`size`	Элемент увеличивается или уменьшается в размерах до указанных значений	`from, origin, restore, scale, to`
`slide`	Элемент появляется/исчезает, въезжая/выезжая из области видимости слева/влево (по умолчанию) или сверху/вверх	`direction, distance`
`transfer`	Элемент перемещается к другому указанному элементу, одновременно изменяясь в размерах, до размеров целевого элемента	`className, to`

Эти эффекты могут использоваться в сочетании с расширенными в jQuery UI версиями функций `show`, `hide` и `toggle` путем передачи им названия желаемого эффекта в первом аргументе. Допускается также передавать дополнительные параметры, изменяющие поведение эффекта, продолжительность воспроизведения и функцию, которая должна быть вызвана по окончании воспроизведения.

```
$('#aDiv').hide('clip');
$('#aDiv').toggle('slide', {direction: 'down'}, 1000);
```

На рис. 10.1 показано несколько примеров эффектов в процессе их воспроизведения.

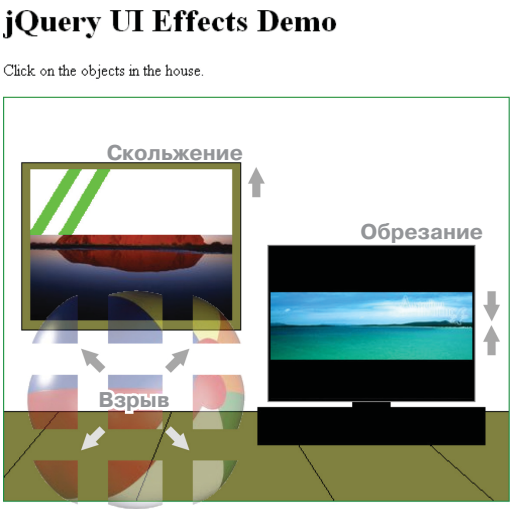


Рис. 10.1 ❖ Демонстрация некоторых эффектов jQuery UI (по часовой стрелке слева сверху): slide (скольжение), clip (обрезание) и explode (взрыв)

Теперь, когда вы узнали, какие эффекты имеются в наличии, пришло время познакомиться с возможностью создания собственных эффектов, которые могли бы использоваться аналогичным способом.

10.2. Добавление нового эффекта

Добавление нового эффекта к существующим в библиотеке jQuery UI заключается в наследовании `$.effects.effect` и добавлении функций, реализующих ваши требования к эффекту. Как и в случае с созданием расширения, чтобы гарантировать надежную работу эффекта, необходимо следовать некоторым принципам.

> В версиях jQuery UI, предшествовавших версии 1.9.0, новый эффект требуется подключать к переменной `$.effects`. В версиях jQuery UI 1.9.0 и выше это и другие требования изменились. Подробнее об этих различиях рассказывается в разделе 10.2.4.

10.2.1. Эффект сжатия

Среди стандартных эффектов в библиотеке jQuery UI имеется эффект с названием `explode`, который «разрывает» указанный элемент на несколько частей, имитируя разлетающиеся осколки, одновременно воспроизводя эффект их плавного растворения. Когда этот же эффект используется для отображения скрытого элемента, происходит собирание элемента из нескольких частей с плавным их проявлением. Для демонстрации приемов создания собственного эффекта мы напишем похожий эффект, имитирующий сжатие, взрыв внутрь элемента. При сокрытии элемента его «осколки» будут не разлетаться вовне, а устремляться к центру, одновременно плавно растворяясь, как показано на рис. 10.2. При отображении скрытого элемента процесс будет протекать в обратном порядке – «осколки будут вылетать из центра к своим местам, одновременно плавно проявляясь, образуя в конечном итоге полный элемент.

В первую очередь необходимо выбрать имя для эффекта, чтобы иметь возможность идентифицировать его (в соответствии с принципом, требующим *ограничиваться единственным именем и использовать только его для взаимодействий с эффектом*), например: `implode`. Реализации существующих эффектов хранятся в файлах с именами, следующими шаблону `jquery.ui.effect-<имя>.js`. Хотя это соглашение не является обязательным, ему часто следуют, чтобы показать, что новый эффект опирается на модуль jQuery UI Effects.

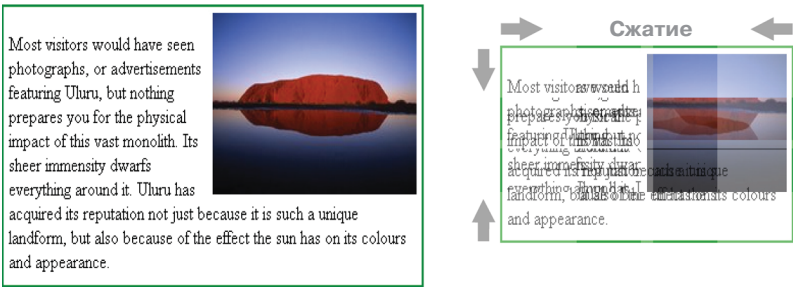


Рис. 10.2 ❖ Эффект сжатия, слева показан исходный элемент, а справа зафиксирован момент на полпути воспроизведения эффекта для его сокрытия

Как обычно, код реализации эффекта следует заключить в анонимную функцию в соответствии с принципами *сокрытия тонкостей реализации с использованием областей видимости* и *отказа от предположения, что имя $ будет ссылаться на jQuery*. Внутри этой функции определяется реализация нового эффекта и осуществляется его интеграция в коллекцию стандартных эффектов библиотеки jQuery (в соответствии с принципом, требующим *помещать все в объект jQuery*). В листинге 10.2 демонстрируется объявление нового эффекта.

Листинг 10.2. Определение эффекта implode для библиотеки jQuery UI

```
(function($) { // Скрывает из видимости внешнего кода,
               // устраняет конфликты с идентификатором $

// ❶ Определение эффекта implode
$.effects.effect.implode = function(options, done) {
    ... // Реализация эффекта
};

})(jQuery);
```

Новый эффект подключается к точке интеграции `$.effects.effect` в виде функции, принимающей два аргумента ❶, в которых передаются параметры настройки (`options`) и функция для запуска воспроизведения последующих эффектов в цепочке (`done`). Аргумент `options` – это объект с атрибутами, хранящими настройки пользователя, которые определяют продолжительность эффекта (`duration`, уже преобразованную в числовую форму, если передавалась пользователем по имени, такому как `slow`), режим работы (`mode`) функции для вызова после завершения воспроизведения анимации (`complete`).

Ваша функция воспроизведения будет вызвана для воспроизведения эффекта в соответствующий момент времени, так как этот эффект может быть частью целой последовательности эффектов. Остальные операции (представленные в листингах 10.3 и 10.4) будут выполняться в контексте данной функции.

10.2.2. Инициализация эффекта

При применении эффекта к целевому элементу вызывается функция, объявленная в листинге 10.2. Прежде чем приступить к реализации самого анимационного эффекта, нам необходимо посмотреть, как следует интерпретировать переданные параметры настройки и инициализировать среду для выполнения последующих операций. В листинге демонстрируется порядок инициализации эффекта.

Листинг 10.3. Инициализация эффекта implode

```
// ❶ Количество фрагментов по умолчанию
var rows = cells = options.pieces ?
    Math.round(Math.sqrt(options.pieces)) : 3;

// ❷ Установить режим показать/скрыть
options.mode = $.effects.setMode($(this), options.mode);
var el = $(this).show().css('visibility', 'hidden');

// ❸ Вычислить смещения
var offset = el.offset();
// Вычесть поля - пока не исправляет проблему
offset.top -= parseInt(el.css('marginTop'), 10) || 0;
offset.left -= parseInt(el.css('marginLeft'), 10) || 0;

var cellWidth = el.outerWidth(true) / cells;
var cellHeight = el.outerHeight(true) / rows;

var segments = $([]);
var remaining = rows * cells;
// ❹ Функция обратного вызова
var completed = function() { // Обратный счетчик до полного завершения
    if (--remaining == 0) {
        options.mode == 'show' ? el.css({visibility: 'visible'}) :
            el.css({visibility: 'visible'}).hide();
        // ❺ Вызвать пользовательскую функцию
        if (options.complete) {
            options.complete.apply(el[0]); // Вызов функции
        }
        // ❻ Выполнить заключительные операции и продолжить
        segments.remove();
        done();
    }
};
```

Работа эффекта начинается с определения значений по умолчанию для параметров настройки ❶. Чтобы сымитировать взрыв внутрь, необходимо разбить элемент на части и каждую из них сместить в центр, поэтому нужно заранее знать, на сколько частей будет разбиваться элемент. С этой целью эффект вычисляет квадратный корень от указанного числа (`options.pieces`) и округляет результат до ближайшего целого, чтобы обеспечить симметричность эффекта. Если число частей не указано, берется значение по умолчанию (в данном случае 3).

Для определения режима работы (показать или скрыть) эффекты jQuery UI используют параметр настройки режима (`options.mode`), и здесь он устанавливается в соответствующее значение `show` или `hide`. Пользователь может также передать значение `toggle`, поэтому тут используется функция `setMode`, которая преобразует его в значение `show` или `hide`, в зависимости от текущего состояния видимости элемента ❷. Свойству CSS `visibility` оригинального элемента присваивается значение `hidden`, чтобы он стал невидимым, но все еще занимал пространство на странице, а ссылка на этот элемент сохраняется в переменной `el` для последующего использования.

В продолжение инициализации вычисляется несколько вспомогательных значений, которые будут использоваться при воспроизведении эффекта ❸, включая смещения оригинального элемента внутри страницы, а также ширину и высоту каждого фрагмента. Вычисляя эти значения здесь, мы избегаем необходимости многократного их вычисления в будущем.

Для выполнения заключительных операций по завершении эффекта определяется функция обратного вызова `completed` ❹. Эта функция будет вызвана для каждого фрагмента, но заключительные операции должны быть выполнены только один раз, поэтому мы уменьшаем счетчик фрагментов, завершивших свое движение (`remaining`), и выполняем операции, только когда он достигнет нуля. В этом случае мы полностью отображаем или скрываем оригинальный элемент, который выше временно был сделан невидимым, в зависимости от установленного режима работы эффекта. Если пользователь указал свою функцию обратного вызова, вызываем эту функцию ❺, передав ей текущий элемент (`el[0]`) в качестве контекста вызова. Удаляем созданные фрагменты элемента и вызываем функцию `done`, чтобы запустить последующие анимационные эффекты, которые могли быть подключены к элементу ❻.

10.2.3. Реализация эффекта

Закончив с подготовительными операциями, можно перейти к следующему шагу – реализации эффекта. На рис. 10.3 показано, как элемент будет разбиваться на фрагменты и как каждый из них будет перемещаться к центру.

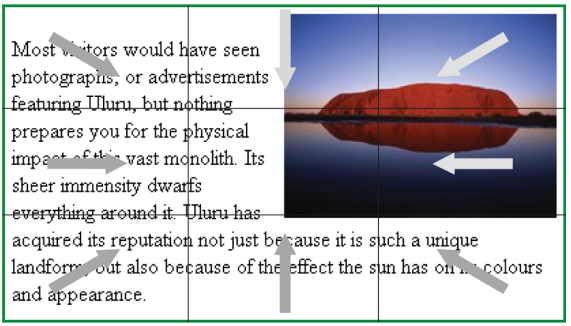


Рис. 10.3 ❖ Элемент разбивается на части, которые затем смещаются по отдельности к центру

В листинге 10.4 представлена реализация эффекта `implode`.

Листинг 10.4. Воспроизведение эффекта implode

```
// ❶ Для каждого фрагмента...
for (var i = 0; i < rows; i++) {
    for (var j = 0; j < cells; j++) {
        // ❷ ...создать копию и обернуть элементом div
        var segment = el.clone().appendTo('body').wrap('<div></div>').
            css({position: 'absolute', visibility: 'visible',
                // ❸ Местоположение внутри контейнера
                left: -j * cellWidth, top: -i * cellHeight}).
            // ❹ Доступ к контейнеру
            parent().addClass('ui-effects-implode').
            // ❺ Местоположение контейнера
            css({position: 'absolute', overflow: 'hidden',
                width: cellWidth, height: cellHeight,
                left: offset.left + j * cellWidth +
                    (options.mode == 'show' ?
                    -(j - cells / 2 + 0.5) * cellWidth : 0),
                top: offset.top + i * cellHeight +
                    (options.mode == 'show' ?
                    -(i - rows / 2 + 0.5) * cellHeight : 0),
                opacity: options.mode == 'show' ? 0 : 1}).
            // ❻ Выполнить перемещение фрагментов посредством
```

```
            //    перемещения контейнеров
            animate({left: offset.left + j * cellWidth +
                    (options.mode == 'show' ? 0 :
                    -(j - cells / 2 + 0.5) * cellWidth),
                top: offset.top + i * cellHeight +
                    (options.mode == 'show' ? 0 :
                    -(i - rows / 2 + 0.5) * cellHeight),
                opacity: options.mode == 'show' ? 1 : 0
            }, options.duration || 500, completed);
        // ❼ Сохранить ссылку на фрагмент
        segments = segments.add(segment);
    }
}
```

Эффект сжатия заключается в разбиении оригинального элемента на фрагменты меньшего размера и смещении их к центру с постепенным растворением. Реализация эффекта выполняет обход всех фрагментов ❶ (`rows` и `cells`) и создает для каждого из них копию оригинального элемента, помещенную в контейнер. При таком подходе появляется возможность показать только часть каждой копии ❷, сместив ее (`left` и `top`) внутри контейнера `div` ❸. В результате появляется несколько элементов `div`, отображающих лишь часть оригинального элемента, каждый из которых можно перемещать независимо друг от друга.

Затем каждый контейнер `div` (`parent`) помечается классом (`ui-effects-implode`), чтобы их проще было идентифицировать ❹ (в функции `completed`, представленной в листинге 10.3). Устанавливается размер контейнера, чтобы в нем отображалась только часть оригинального элемента, и определяется его начальное местоположение ❺. Когда эффект используется для сокрытия элемента, все контейнеры будут располагаться в позициях, соответствующих отображаемым в них фрагментам оригинального элемента, которые определяются по текущим индексам (`i` и `j`) и размерам. Когда эффект используется для отображения прежде скрытого элемента, все фрагменты первоначально помещаются в центр друг над другом и затем перемещаются к соответствующим местоположениям, чтобы в конце получился полностью собранный элемент.

В заключение запускается анимация координат контейнера `div` от начального до конечного значений ❻. По завершении анимации каждого фрагмента вызывается функция `completed` из листинга 10.3, которая в конечном итоге выполнит заключительные операции и удалит фрагменты, созданные здесь ❼.

10.2.4. Реализация эффекта в версиях jQuery UI ниже 1.9

Порядок реализации эффектов изменился в версии jQuery UI 1.9.0, и необходимо внести в программный код некоторые изменения, чтобы он смог работать с более ранними версиями библиотеки. К счастью, большая часть кода может использоваться повторно, как показано в листинге 10.5.

Листинг 10.5. Реализация эффекта implode в более ранних версиях jQuery UI

```
// ❶ Проверить версию
var newEffects = !!$.effects.effect; // Использовать новую инфраструктуру?

// ❷ Если доступна новая версия, объявить эффект для jQuery UI 1.9.0
if (newEffects) {
    $.effects.effect.implode = function(options, done) {
        // ❸ Вызвать функцию эффекта
        implodeIt.apply(this, arguments);
    };
}
else {
    // ❹ Иначе объявить эффект для более ранних версий jQuery UI
    $.effects.implode = function(o) {
        // ❺ Преобразовать параметры
        var options = $.extend({complete: o.callback}, o, o.options);
        // ❻ Поставить эффект в очередь для обработки
        return this.queue(function() {
            var el = $(this);
            // ❼ Вызвать функцию эффекта
            implodeIt.apply(this, [options, function() {
                el.dequeue();
            }]);
        });
    };
}

/* Немедленно применяет эффект implode.
   @param  options  (object) настройки эффекта
   @param  done     (function) функция для вызова по завершении эффекта */
// ❽ Извлеченный эффект функции
function implodeIt(options, done) {
    ... // Тот же код, что и для версии jQuery UI 1.9.0
}
```

Сначала необходимо определить, старая или новая версия инфраструктуры будет использоваться, что достигается проверкой наличия точки интеграции `$.effects.effect` ❶. Если искомая точка при-

сутствует, новый эффект объявляется, как было показано выше ❷, но в этом случае вызывается общая функция, реализующая фактический эффект ❸.

Если используется инфраструктура более ранней версии, эффект должен подключаться к точке интеграции `$.effects` под тем же именем, но в этой версии функции эффекта будет передаваться только один аргумент (`o`) ❹, содержащий параметры настройки. Параметры настройки структурированы иначе, чем в версии jQuery 1.9.0, поэтому их необходимо преобразовать в формат, применяемый в более новой версии, чтобы использовать общую реализацию функции ❺. Новая структура имеет более плоскую организацию, поэтому пользовательские настройки (`o.options`) следует перенести туда, где уже находятся настройки по умолчанию. Кроме того, ссылка на функцию обратного вызова, определяемую пользователем, в ранних версиях хранилась в атрибуте `callback`, и ее нужно сохранить в другом атрибуте (`complete`).

В старых версиях функция эффекта должна возвращать ссылку на коллекцию jQuery, для чего вызывается функция `queue` ❻. Вызов `queue` добавит функцию обратного вызова в стандартную очередь `fx`, имеющуюся у каждого выбранного элемента, чтобы запустить воспроизведение эффекта, когда до него дойдет очередь. В jQuery 1.9.0 манипуляции с очередями выполняются без вашего участия, благодаря чему это требование было устранено в новой версии.

Функция, поставленная в очередь, вызывает общую функцию реализации эффекта, передавая ей преобразованные параметры и функцию `done`, которая запускает обработку следующего элемента в стандартной очереди `fx` элемента вызовом функции `dequeue` ❼.

Общая функция реализации эффекта определена, как для версии jQuery 1.9.0, с двумя аргументами ❽, и содержит тот же самый код, что и функция, представленная ранее.

Код из листинга 10.5 обеспечивает возможность использования эффекта во всех версиях jQuery без каких-либо усилий со стороны пользователя.

10.2.5. Законченный эффект

На этом реализацию нового эффекта jQuery UI, скрывающего или отображающего элемент путем сжатия или расширения в/из центра, можно считать завершенной. Весь программный код эффекта можно найти на веб-сайте книги.

Чтобы задействовать новый эффект, необходимо подключить к странице обе библиотеки – jQuery и jQuery UI (хотя бы модуль

Effects), а также сам эффект. Затем применить эффект к элементу, указав его имя в вызове функции `show`, `hide` или `toggle`. На этом этапе можно также передать дополнительные параметры настройки.

```
$('#aDiv').hide('implode');
$('#aDiv').toggle('implode', {pieces: 25}, 1000,
    function() { alert('Done'); });
```

По умолчанию действие механизма анимации заключается в равномерном изменении значений атрибутов на протяжении некоторого отрезка времени. Однако существуют другие способы изменения значений для достижения интересных и более реалистичных движений. Об этом рассказывается далее.

10.3. Функции управления переходами

Суть механизма анимации заключается в изменении атрибута от одного значения до другого за определенный интервал времени, что вызывает соответствующее изменение внешнего вида элемента. Но равномерное изменение значения не всегда соответствует нашим ожиданиям, так как мы склонны проводить аналогии между элементами на странице и реальными объектами в физическом мире. К счастью, библиотека jQuery поддерживает возможность таких нелинейных изменений, позволяя добиваться дополнительных анимационных эффектов.

10.3.1. Что такое «функция управления переходом»?

В реальном мире влияние на скорость и ускорение физических тел могут оказывать разные силы, такие как сила тяжести и сила трения. В компьютерном мире скорость и ускорение изменения значения атрибута зависит от *функции управления переходом* (easing).

Под *ускорением* понимается увеличение скорости. Под *замедлением* – уменьшение, вплоть до остановки. Иногда ускорение и замедление комбинируются для получения кривой изменения скорости, когда движение начинается с небольшой скорости, потом скорость увеличивается, а затем уменьшается до остановки.

Функция управления переходом – это обычная функция, возвращающая значение атрибута для данной отметки времени. Чтобы уложиться в разные интервалы времени, отводимые пользователем для воспроизведения эффектов, и диапазоны представления значений атрибутов, функция получает нормализованное значение вре-

мени в интервале от 0.0 до 1.0, где 0.0 соответствует моменту начала воспроизведения эффекта, а 1.0 – конца, и должна возвращать также нормализованное значение для атрибута в диапазоне от 0.0 до 1.0. Например, в листинге 10.6 представлена функция `swing`.

Листинг 10.6. Функция управления переходом swing

```
jQuery.easing = {
    ...
    swing: function( p ) {
        return 0.5 - Math.cos( p*Math.PI ) / 2;
    }
};
```

> Принцип действия функций управления переходами изменился в jQuery 1.7.2 и jQuery UI 1.8.23, как описывается в данном разделе. В предшествующих версиях функции управления переходами отвечали за вычисление фактического значения атрибута, соответствующего конкретному моменту времени, и получали в качестве аргументов начальное значение атрибута и разницу между начальным и конечным значениями. Далее в этой книге будут обсуждаться только новые версии.

Чтобы задействовать функцию управления переходом в воспроизведении анимационного эффекта, необходимо передать ее имя в качестве одного из аргументов. Например, чтобы изменить высоту элемента с пружинящим эффектом, можно воспользоваться функцией `easeInOutElastic`:

```
$('#mydiv').animate({height: 200}, 1000, 'easeInOutElastic');
```

Для одного и того же анимационного эффекта к разным атрибутам можно применять разные функции переходов. Например, можно задействовать функцию `easeInOutElastic` для всех атрибутов, кроме цвета фона, и изменять последний по линейному закону:

```
$('#mydiv').animate({height: 200, backgroundColor: ['red', 'linear']},
    1000, 'easeInOutElastic');
```

Проще всего понять влияние той или иной функции перехода – взглянуть на ее график. На рис. 10.4 изображен график функции `swing`. Время увеличивается слева направо, а значение атрибута изменяется от начального до конечного – снизу вверх. В данном случае можно видеть, что сначала значение атрибута изменяется медленно, затем скорость изменения увеличивается и потом опять замедляется. Соответственно, это комбинированная функция управления переходом.

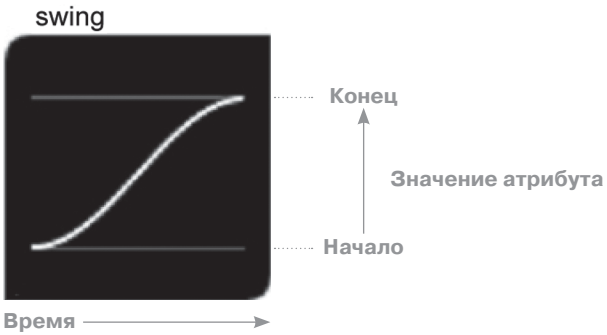


Рис. 10.4 ❖ График функции перехода, демонстрирующий изменение значения атрибута с течением времени

10.3.2. Существующие функции управления переходами

Непосредственно в библиотеке jQuery определены две функции перехода, `linear` и `swing`, вторая из которых используется по умолчанию. Функции управления переходами, предлагаемые библиотекой jQuery UI, сосредоточены в расширении jQuery Easing (http://gsgd.co.uk/sandbox/jquery/easing/), которое, в свою очередь, входит в состав модуля jQuery UI Effects. Эти функции являются JavaScript-версиями функций Роберта Пеннера (Robert Penner) (http://www.robertpenner.com/easing/). Стандартная функция `swing` в этом расширении переименована в `jswing`, а по умолчанию используется функция `easeOutQuad`.

На рис. 10.5 показаны графики функций переходов из библиотек jQuery и jQuery UI.

Функция `linear` определяет постоянную скорость изменения атрибута от начального до конечного значения, тогда как функция `swing` опирается на косинусоиду и определяет небольшое ускорение в начале и соответствующее замедление в конце. Следующие далее функции (от `easeInQuad` до `easeInOutExpo`) определяют более высокие ускорения и замедления, основываясь на математических функциях прогрессий (квадратичной, кубической, пятой степени и экспоненциальной). В каждой группе имеется версия, действующая только с ускорением, только с замедлением и комбинированная.

Следующие далее функции основаны на синусоиде и окружности. Функции `elastic` и `back` интересны тем, что для достижения своего

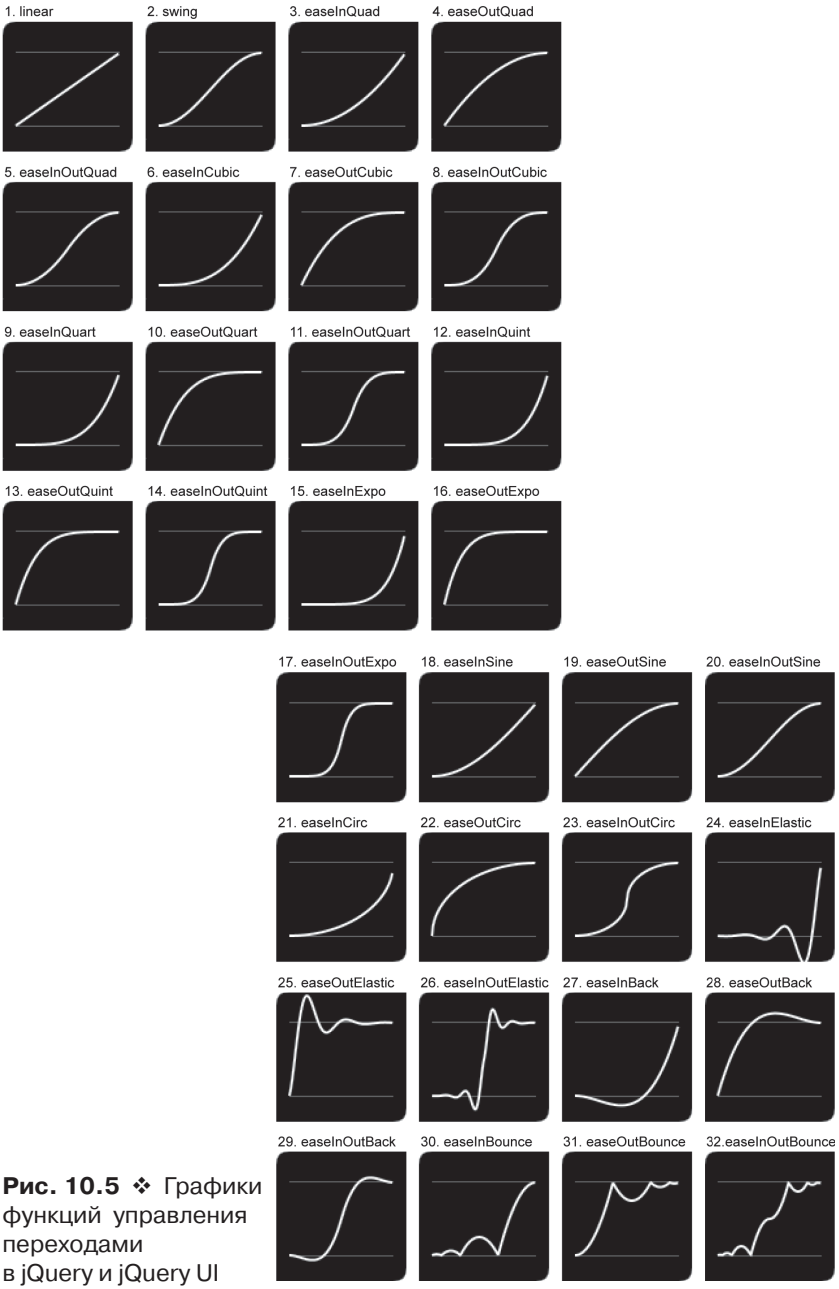


Рис. 10.5 ❖ Графики функций управления переходами в jQuery и jQuery UI

эффекта они изменяют значение атрибута так, что оно выходит за пределы установленного диапазона. Функции семейства `elastic` реализуют колебание значения атрибута вокруг начального и конечного значений, которые напоминают колебания пружины, а функции семейства `back` определяют лишь один кратковременный выход за начальную и конечную границы диапазона. Функции семейства `bounce` имитируют нарастающий или затухающий эффект подпрыгивающего мячика. Все эти семейства имеют версию, действующую только с ускорением, только с замедлением и комбинированную.

10.3.3. Добавление новой функции управления переходом

С помощью функций управления переходами можно воспроизводить некоторые эффекты для привлечения внимания, как, например, эффект `bounce`, упоминавшийся в предыдущем разделе. Чтобы придать своему эффекту неповторимые черты, мы определим собственные функции управления переходами. Один такой эффект заключается в том, чтобы на полпути возвращать значение атрибута немного назад и тем самым сделать стандартную анимацию немного интереснее (см. функцию `bump` ниже).

Добавление собственных функций управления переходами осуществляется путем их подключения к точке интеграции `$.easing`. Функция должна принимать один аргумент, представляющий долю времени, истекшую с момента начала воспроизведения эффекта (от 0.0 до 1.0), и возвращать пропорцию разницы между начальным и конечным значениями атрибута (от 0.0 в момент начала эффекта до 1.0 в момент окончания). Библиотека jQuery сама преобразует эту разницу в фактическое значение атрибута.

На рис. 10.6 изображены графики двух стандартных функций управления переходами, `linear` и `swing`, а также несколько нестандартных, реализация которых приводится в листинге 10.7.

Листинг 10.7. Собственные функции управления переходами

```
/* ❶ Функция перехода bump. */
$.easing.bump = function(p) {
    return (p < 0.5 ? Math.sin(p * Math.PI * 1.46) * 2 / 3 :
        1 - (Math.sin((1 - p) * Math.PI * 1.46) * 2 / 3));
};

/* ❷ Функция перехода zigzag. */
$.easing.zigzag = function(p) {
```

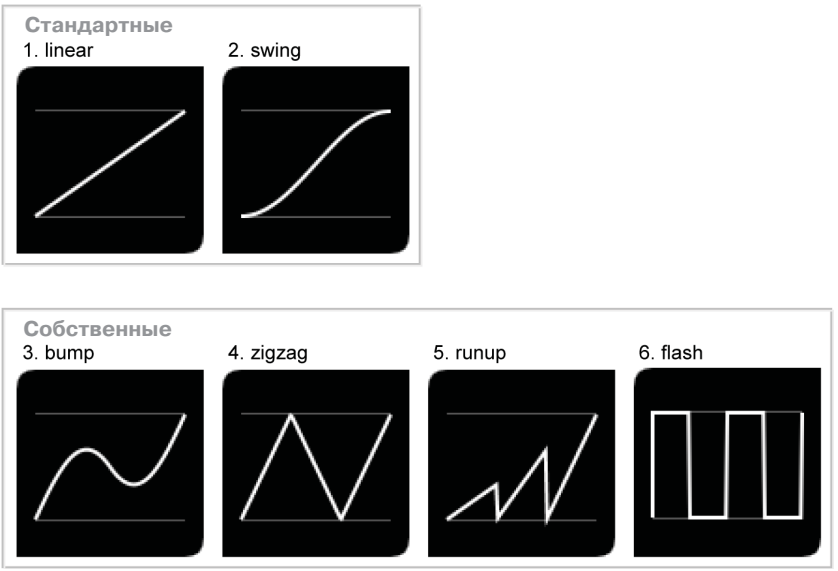


Рис. 10.6 ❖ Графики стандартных и собственных функций перехода

```
    return 3 * (p < 0.333 ? p : (p < 0.667 ? 0.667 - p : p - 0.667));
};

/* ❸ Функция перехода runup. */
$.easing.runup = function(p) {
    return (p < 0.333 ? p :
        (p < 0.667 ? (p - 0.333) * 2 : (p - 0.667) * 3));
};

/* ❹ Функция перехода flash. */
$.easing.flash = function(p) {
    return Math.floor(p * 4 + 1) % 2;
};
```

Функция перехода `bump` ❶ изменяет значение атрибута, сначала приближая его к конечному значению, затем немного возвращая назад, и затем продолжает его изменение в сторону конечного значения. Для достижения описанного эффекта она объединяет две синусоиды. Функция перехода `zigzag` ❷ изменяет значение атрибута линейно, до его конечной величины, затем так же линейно возвращает его в начальное значение и потом вновь изменяет до конечного, при этом каждый переход между начальным и конечным значениями длится ровно одну треть от общей продолжительности эффекта.

Функция перехода runup ❸ изменяет значение атрибута в три этапа – линейно приближая все ближе к конечному значению и мгновенно возвращая в исходное состояние на каждом этапе. Функция перехода flash ❹ также выполняет изменение атрибута в три этапа: она мгновенно переключает атрибут между начальным и конечным значениями, выдерживает паузу и производит обратное переключение, создавая тем самым эффект «моргания».

Имена этих функций управления переходами можно передавать непосредственно в аргументе функции animate:

```
$('#aDiv').animate({height: 300}, 1000, 'runup');
```

либо в паре с анимируемыми атрибутами, после конечного значения:

```
$('#aDiv').animate({height: [300, 'bump'], opacity: 0.0}, 1000, 'runup');
```

Полный код реализации этих функций управления переходами доступен для загрузки на веб-сайте книги.

Функции управления переходами в ранних версиях jQuery

Чтобы обеспечить поддержку всех версий jQuery, необходимо определить основную функцию перехода отдельно. В новых версиях jQuery (если $.support.newEasing имеет значение true) следует указывать прямую ссылку на эту функцию, а в старых версиях – использовать функцию-обертку, вычисляющую фактическое значение с использованием основной функции перехода, начального значения атрибута и разницы между ним и конечным значениями.

```
$.support.newEasing = ($.easing.linear(1.0) == 1.0);

function bumpEasing(p) {
    return (p < 0.5 ? Math.sin(p * Math.PI * 1.46) * 2 / 3 :
        1 - (Math.sin((1 - p) * Math.PI * 1.46) * 2 / 3));
}

if ($.support.newEasing) {
    $.easing.bump = bumpEasing;
}
else {
    $.easing.bump = function(p, n, firstNum, diff) {
        return firstNum + diff * bumpEasing(p);
    };
}
```

Это нужно знать

Создавайте собственные эффекты для придания неповторимых визуальных особенностей своим веб-страницам.

Библиотека jQuery UI включает эффекты, изменяющие внешний вид элементов при их появлении или исчезновении, а также с целью привлечь внимание пользователя.

Библиотека jQuery UI предоставляет общую функциональность, используемую всеми эффектами.
Добавление новых эффектов выполняется путем подключения их к точке интеграции `$.effects.effect` (`$.effects` в версиях jQuery UI ниже 1.9).
Функции переходов управляют скоростями изменения значений свойств. Добавление новых функций перехода выполняется путем подключения их к точке интеграции `$.easing` (но в версиях библиотек, предшествующих версиям jQuery 1.7.2 и jQuery UI 1.8.23, они реализуются несколько иначе).

Попробуйте сами

Создайте новый эффект с именем `spiral`. Он должен разбивать целевой элемент на четыре части, после чего левый верхний фрагмент должен смещаться вниз, правый верхний – влево, правый нижний – вверх, и левый нижний – влево. По мере смещения каждый фрагмент должен растворяться до полного исчезновения. При отображении элемента направление движения фрагментов меняется на обратное.
Добавьте новую функцию управления переходом, которая превышает конечное значение на 10% перед переходом к нему. Чтобы усложнить задачу, реализуйте управление изменением не по линейному, а по параболическому закону.

10.4. В заключение

Модули, входящие в состав библиотеки jQuery UI, включают множество вспомогательных функций, низкоуровневых механизмов (таких как «перетащил и бросил»), высокоуровневых компонентов и виджетов (таких как Tabs и Datepicker), а также визуальных эффектов. Вы можете использовать эти эффекты для улучшения визуального представления элементов или привлечения внимания пользователей к данным элементам. Чтобы помочь вам в создании собственных эффектов, в библиотеку включено множество функций общего пользования.

Собственные эффекты, действующие наравне со встроенными, следует подключать к точке интеграции `$.effects.effect`. Например, в этой главе была описана реализация эффекта `implode`, дополняющего существующий эффект `explode`, который выражается в сжатии элемента вовнутрь.

Базовая библиотека jQuery также предоставляет возможность изменять скорость и ускорение изменения атрибута с течением времени посредством применения функции управления переходом. В ней определяются всего две такие функции, но jQuery UI добавляет целое множество подобных функций, позволяя добиваться интересных анимационных эффектов. Разумеется, вы можете добавить собственные

функции управления переходами, чтобы определить новые способы перехода значений атрибутов из начального в конечное состояние.

На этом мы завершаем наше знакомство с библиотекой jQuery UI, ее модулями и точками интеграции. В следующей, последней части книги вы познакомитесь с другими аспектами применения библиотеки jQuery, доступными для расширения, и начнем мы с анимации атрибутов, содержащих сложные, составные значения.

Часть IV

Прочие расширения

В библиотеке jQuery имеется еще несколько точек интеграции, с которыми мы познакомимся в этой части.

jQuery способна автоматически производить анимационные эффекты на основе простых числовых значений, но более сложные или составные значения библиотекой не поддерживаются. В главе 11 мы посмотрим, как добавить поддержку анимации для этих и других типов значений для использования совместно с существующим механизмом анимационных эффектов.

Возможность использовать технологию Ajax для извлечения удаленного содержимого и его обработки без полной перезагрузки страницы отлично поддерживается библиотекой jQuery. В главе 12 демонстрируется, как можно внедриться в работу механизма поддержки Ajax с целью расширения встроенных возможностей, от предварительной обработки запросов до подстановки альтернативных механизмов удаленных взаимодействий и преобразования полученных данных в более удобные форматы.

Обработка событий – еще один аспект веб-разработки, который упрощается библиотекой jQuery за счет сокрытия различий между браузерами. В jQuery реализована специальная инфраструктура поддержки событий, описываемая в главе 13, которая позволяет добавлять новые события или изменять порядок обработки существующих.

Наконец, в главе 14 обсуждается добавление новых правил проверки в расширение Validation. И хотя это расширение не является частью библиотеки jQuery, оно получило весьма широкое распространение и имеет собственные точки интеграции.

Глава 11

Анимация свойств

Эта глава охватывает следующие темы:

- инфраструктура поддержки анимационных эффектов в библиотеке jQuery;
- добавление нового эффекта для анимации свойства.

Одной из самых широко используемых особенностей jQuery является механизм анимации, позволяющий изменять различные свойства элементов и тем самым воспроизводить визуальные анимационные эффекты. Помимо простых функций `show` и `hide`, реализующих простые переходы между состояниями видимости, если указана продолжительность, существует также несколько стандартных анимационных эффектов, воспроизводящих сдвиг и растворение/проявление, таких как `slideDown` и `fadeIn`. Если требуется что-то более экзотическое, можно определить собственный анимационный эффект перемещения элемента в определенную позицию или изменения его размеров с использованием функции `animate`.

```
$('#myDiv').slideDown('slow');
$('#myDiv').animate({width: '20%', left: '100px'});
```

Но встроенные функции анимации могут обслуживать только свойства с простыми значениями: числовыми, за которыми может следовать обозначение единиц измерения, такие как `200` (пиксели), `2em` или `50%`. Чтобы обеспечить поддержку более сложных значений свойств, таких как цвет (`#CCFFCC`), необходимо реализовать собственную функцию, которая будет знать, как интерпретировать сложное значение, выполнять изменение значения в процессе анимации и присваивать новое значение свойству.

Библиотека jQuery UI включает собственные анимационные эффекты для поддержки свойств, содержащих значения цвета, таких как `color` и `background-color`. Эти функции знают, как интерпретировать значения цвета в форматах, поддерживаемых CSS, которые могут определяться в шестнадцатеричном виде, в виде триад RGB или

даже названий. На каждом шаге анимации они извлекают из значений цветов величину красной, зеленой и синей составляющей, изменяют эти составляющие по отдельности и затем компонуют их обратно в единое значение цвета. Наличие поддержки анимации цвета позволяет указывать значения цвета наряду с обычными числовыми значениями.

```
$('#myDiv').animate({fontSize: '20px', backgroundColor: '#DDFFE8'});
```

Чтобы получить возможность анимировать свойства с нестандартными значениями, необходимо найти соответствующее расширение или написать свое, как описывается в этой главе. Как только у вас появится такое расширение, библиотека будет интерпретировать подобные свойства как стандартные, позволяя применять к ним механизм анимации, как обычно. Например, вы сможете применять собственные функции управления переходами или передавать свои функции для вызова по окончании анимации.

В этой главе мы реализуем поддержку анимации для свойства стиля `background-position`, чтобы показать, как обрабатывать сложные значения и как учитывать различия между механизмами анимации в разных версиях jQuery.

11.1. Инфраструктура поддержки анимационных эффектов

Функция `animate` из библиотеки jQuery воспроизводит анимационные эффекты на основе одного или нескольких свойств выбранных элементов – достаточно передать ей список свойств, подлежащих изменению, и конечные значения для каждого. После этого механизм анимации будет постепенно изменять значения указанных свойств от начального к конечному, оказывая тем самым влияние на внешний вид элементов веб-страницы.

Обратите внимание, что jQuery накладывает некоторые ограничения на свойства, которые могут анимироваться. За исключением свойств, хранящих простые числовые значения, jQuery не может интерпретировать свойства, значения которых имеют форму стандартных имен. Например, свойство, хранящее толщину рамки элемента, должно иметь числовое значение, а не именованное, такое как `thin` или `thick`. Кроме того, не допускается смешивать разные единицы измерения числовых свойств, то есть нельзя начать анимацию свойства,

хранящего значение в пикселях, а заканчивать значением процентов, потому что jQuery не предусматривает автоматического преобразования между единицами измерения.

В следующих разделах описываются встроенные возможности механизма анимации jQuery, такие как установка продолжительности анимации, использование альтернативных функций управления переходами для создания дополнительных эффектов и вызов пользовательской функции по завершении анимации, а также будут представлены внутренние механизмы библиотеки, решающие эти задачи.

11.1.1. Механизм анимации

Некоторые наиболее часто используемые эффекты реализованы в библиотеке jQuery в виде отдельных функций, включающих в процесс анимации сразу несколько взаимосвязанных свойств. Если передать стандартным функциям `show`, `hide` или `toggle` продолжительность эффекта, целевой элемент будет проявляться или растворяться и разворачиваться вниз и вправо из точки в левом верхнем углу, как показано на рис. 11.1. Аналогично функции `fadeIn`, `fadeOut` и `fadeToggle` отображают или скрывают элемент, изменяя его прозрачность, а функции `slideDown`, `slideUp` и `slideToggle` изменяют высоту элемента.

При попытке одновременно применить несколько анимационных эффектов к одному и тому же элементу они по умолчанию будут помещены в очередь (с именем `fx`) и выполнены последовательно, друг за другом. Однако есть возможность запускать анимационные эффекты параллельно, установив параметр настройки `queue` в значение `false`.

Новые значения свойств могут задаваться в виде единственного числа (в этом случае оно означает «пиксели») или числа с единицами измерения, например `2em` или `50%`. Кроме того, можно определить изменение относительно текущего значения, указав перед новым значением оператор `-=` или `+=`, в результате чего конечное значение будет определено как разность или сумма текущего и указанного значений.

В вызов функции `animate` можно также передавать дополнительные параметры, влияющие на ее поведение.

```
$('#myDiv').animate({left: '50px', width: '-=50px'},
    {duration: 500, easing: 'linear', complete: function() {
        $('#myDivController').attr('src', 'img/expand.gif');
}});
```

Параметр `duration` определяет продолжительность анимационного эффекта. В нем допускается передавать число (миллисекунды)

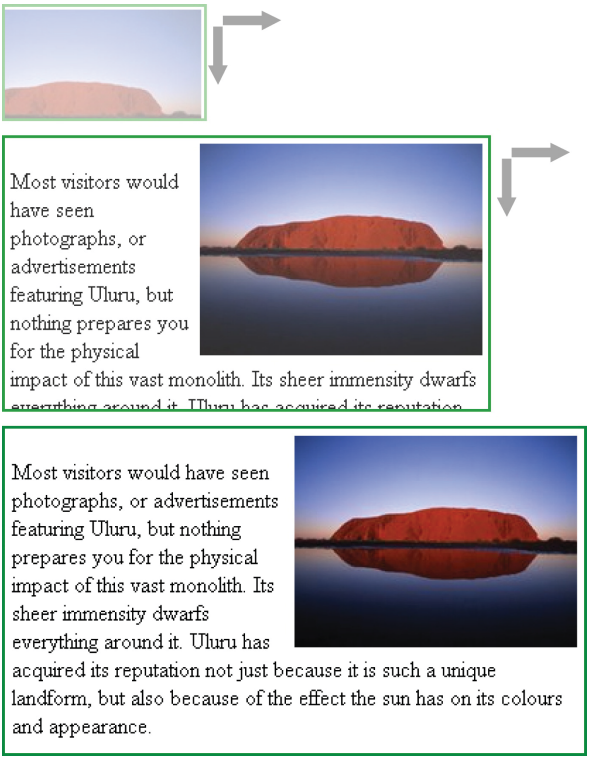


Рис. 11.1 ❖ Процесс воспроизведения анимационного эффекта, производимого функцией show() (сверху вниз): 35%, 85% и 100% от общей продолжительности эффекта

или имя (`slow`, `normal` или `fast`). Параметр `easing` определяет функцию управления изменением значения свойства с течением времени (функции управления переходами подробно рассматривались в главе 10). Начиная с версии jQuery 1.4, появилась возможность определять функции перехода отдельно для каждого свойства, что обеспечивает возможность более тонкого управления анимационными эффектами.

Чтобы получить извещение об окончании анимационного эффекта, в параметре `complete` можно передать ссылку на функцию обратного вызова. Она будет вызвана один раз для каждого элемента, участвующего в анимации по окончании воспроизведения эффекта, независи-

мо от количества изменяемых свойств. Функции обратного вызова не будет передано никаких аргументов, но переменная `this` в ней будет ссылаться на текущий элемент.

В параметре `step` можно передать свой обработчик для вызова на каждом шаге воспроизведения анимационного эффекта. Она должна принимать два аргумента, текущее значение свойства (`now`) и объект с информацией о свойствах, подлежащих изменению (`tween`), а в переменной `this` – ссылку на текущий элемент. Функция будет вызываться по одному разу для каждого свойства каждого элемента на каждом шаге анимации, что позволяет следить за процессом и, при необходимости, вмешиваться в него.

Но что происходит в недрах jQuery при вызове `animate`? Об этом рассказывается дальше.

11.1.2. Порядок выполнения анимации

Для управления процессом анимации библиотека jQuery использует объекты `Deferred`, которые, в свою очередь, опираются на стандартную JavaScript-функцию `setInterval`, приостанавливающую выполнение кода. По умолчанию величина задержки составляет 13 миллисекунд и определяется значением переменной `$.fx.interval`. На каждом шаге анимации jQuery вызывает функцию, реализующую изменение значений свойств. Имейте в виду, что на каждом шаге jQuery вычисляет время, затраченное на выполнение этого шага, а не полагается на простой цикл, что позволяет ей обеспечить более точное воспроизведение эффектов.

В версиях jQuery ниже 1.8.0 механизм анимации не использовал объекты `Deferred`, а вызывал функцию `setInterval` непосредственно.

Объекты Deferred

В jQuery 1.5 появилась функция `jQuery.Deferred()`, возвращающая вспомогательный объект, который можно использовать для регистрации множества очередей функций обратного вызова, вызова этих очередей. Этот объект в своей работе опирается на состояние успеха или неудачи вызовов любых синхронных или асинхронных функций (http://api.jquery.com/category/deferred-object/)[1].

Функция `jQuery.Deferred()` привнесла более совершенные приемы управления функциями обратного вызова и их вызова. В частности, `jQuery.Deferred()` предоставляет гибкую возможность зарегистрировать несколько функций, которые будут вызваны при выполнении некоторого условия или незамедлительно, если это условие уже выполнено.

---

[1] http://jquery.page2page.ru/index.php5/Объект_deferred – *Прим. перев.*

Функции-обработчики, вызываемые в ходе анимации, хранятся в объекте `$.Tween.propHooks` и индексируются именами свойств. Если пользователь не указал собственный обработчик, используется обработчик по умолчанию `_default`, представленный в листинге 11.1. Этот обработчик понимает только простые числовые значения и единицы измерения и может применяться к свойствам самих элементов, а также к свойствам стилей, установленных вызовом функции `css`.

Листинг 11.1. Обработчик анимации по умолчанию

```
// ❶ Точка интеграции для подключения своей анимации
Tween.propHooks = {
    // ❷ Обработчик значений свойств по умолчанию
    _default: {
        // ❸ Извлекает значение свойства
        get: function( tween ) {
            var result;

            // ❹ Получить значение атрибута
            if ( tween.elem[ tween.prop ] != null &&
                    (!tween.elem.style ||
                    tween.elem.style[ tween.prop ] == null)){
                return tween.elem[ tween.prop ];
            }

            // передача любого значения функции .css в 4 параметре
            // автоматичеки приведет к попытке вызова parseFloat и
            // возврата к строковому представлению, если попытка
            // окажется неудачной, поэтому простые значения, такие как
            // "10px", будут преобразованы функцией Float. Сложные
            // значения, такие как "rotate(1rad)", возвращаются как есть.
            // ❺ Получить значение стиля
            result = jQuery.css( tween.elem, tween.prop, false, "" );
            // Пустые строки, null, undefined и "auto"
            // преобразуются в 0.
            return !result || result === "auto" ? 0 : result;
        },
        // ❻ Устанавливает значение свойства
        set: function( tween ) {
            // использовать функцию step для обратной совместимости -
            // использовать cssHook, если определена -
            // использовать .style, если определена,
            // и использовать простые свойства, если доступны
            // ❼ Резервный вариант для версий jQuery ниже 1.8
            if ( jQuery.fx.step[ tween.prop ] ) {
                jQuery.fx.step[ tween.prop ]( tween );
            // ❽ Изменить стиль
            } else if ( tween.elem.style && ( tween.elem.style[
                    jQuery.cssProps[ tween.prop ]] != null ||
```

```
                    jQuery.cssHooks[ tween.prop ] ) ) {
                jQuery.style( tween.elem, tween.prop,
                    tween.now + tween.unit );
            // ❾ Изменить атрибут
            } else {
                tween.elem[ tween.prop ] = tween.now;
            }
        }
    }
};
```

Обработчик по умолчанию (с именем `_default` ❷) определяется как часть объекта `Tween.propHooks` ❶ (и позднее используется под псевдонимом `$.Tween.propHooks`). Каждый обработчик – это объект с двумя атрибутами: функция чтения (`get`) ❸, используемая для извлечения текущего значения свойства, и функция записи (`set`) ❻, используемая для записи нового значения.

> Порядок определения обработчиков анимации изменился в версии jQuery 1.8.0. Прежде можно было определить единственную функцию записи, сохранив ссылку на нее в свойстве `$.fx.step`. В следующих разделах основное внимание будет уделяться новым версиям, тем не менее мы познакомимся с особенностями определения обработчиков анимации в более ранних версиях jQuery.

Параметр, что передается функциям чтения и записи (`tween`), содержит информацию о текущем анимационном эффекте. В нем имеются ссылка на текущий элемент (`tween.elem`), имя свойства, подлежащего анимации (`tween.prop`) ❹ ❾, и имя функции управления переходом (`tween.easing`) для данного свойства. В нем также имеется объект (`tween.options`), хранящий параметры настройки, переданные в вызов функции `animate`, и значения по умолчанию, если таковые имеются. В число параметров входят: продолжительность эффекта (`duration`), функция управления переходом по умолчанию (`easing`), а также ссылки на функции обратного вызова `complete` и `step`.

Функция чтения пытается получить значение свойства CSS ❺, прежде чем преобразовать неизвестное значение в 0.

Функции записи ❻ в аргументе `tween` передается дополнительная информация о текущем шаге анимации, в том числе начальное (`tween.start`) и конечное (`tween.end`) значения свойства, а также текущее значение (`tween.now`) ❽ – все в виде простых числовых значений. Единицы измерения этих значений хранятся в атрибуте `tween.unit`, а в атрибуте `tween.pos` хранится дробное значение в диапазоне от 0 до 1, определяющее истекшее время воспроизведения эффекта.

Для обратной совместимости функция проверяет свойство `$.fx.step` ❼ и вызывает указанную в нем функцию. В противном случае вычисляет текущее значение для данного шага анимации как произведение доли истекшего времени на разность между конечным и начальным значениями плюс начальное значение.

```
tween.pos * (tween.end - tween.start) + tween.start
```

Перед записью вычисленного значения обратно в указанное свойство к нему присоединяются единицы измерения.

Поиск обработчика анимации выполняется по имени свойства, подлежащего анимации, а его функции чтения и записи вызываются в соответствующие моменты времени. Всем процессом – вычисления начального значения, разности конечного и начального значений, планирования регулярных обновлений свойств, вызова обработчика на каждом шаге, выполнения заключительных операций и вызова функции пользователя по завершении (при необходимости) – управляет сама библиотека jQuery.

Чтобы разобраться со всеми тонкостями, мы создадим собственный обработчик анимации для обслуживания свойства стиля `background-position`, значение которого не является простым числом.

11.2. Добавление собственного обработчика анимации свойства

Многие свойства имеют значения, которые с успехом могут обрабатываться встроенными функциями, такие как простые числовые значения с необязательным спецификатором единиц измерения. Однако иногда может потребоваться воспроизвести эффект на основе других свойств, значения которых не укладываются в этот формат. Не зная, как интерпретировать и изменять такие свойства, jQuery не сможет воспроизвести анимационный эффект на их основе. Обработчик анимации для свойства `background-position`, описываемый в следующем разделе, обслуживает составные значения, состоящие из двух чисел с единицами измерения, и позволяет обеспечить плавное изменение обоих за определенный интервал времени.

Собственные обработчики анимации должны подключаться к точке интеграции `$.Tween.propHooks` и определять функции чтения и записи значений требуемого формата. После подключения jQuery сможет использовать эти обработчики в стандартной процедуре обработки анимации для изменения значений соответствующих свойств.

11.2.1. Анимация свойства background-position

Свойство `background-position` – это свойство CSS, имеющее нестандартный формат. Его значение состоит из двух чисел, определяющих смещение от верхнего левого угла элемента фонового изображения отдельно по вертикали и по горизонтали (если задано два числа) или одновременно по вертикали и горизонтали (если задано одно число). Изменяя позицию фонового изображения, можно добиться интересных эффектов, таких как прокрутка фонового изображения ландшафта или выделение каких-либо элементов изображения, повышая привлекательность веб-сайта для посетителей. Пример одного из таких эффектов приводится на рис. 11.2.

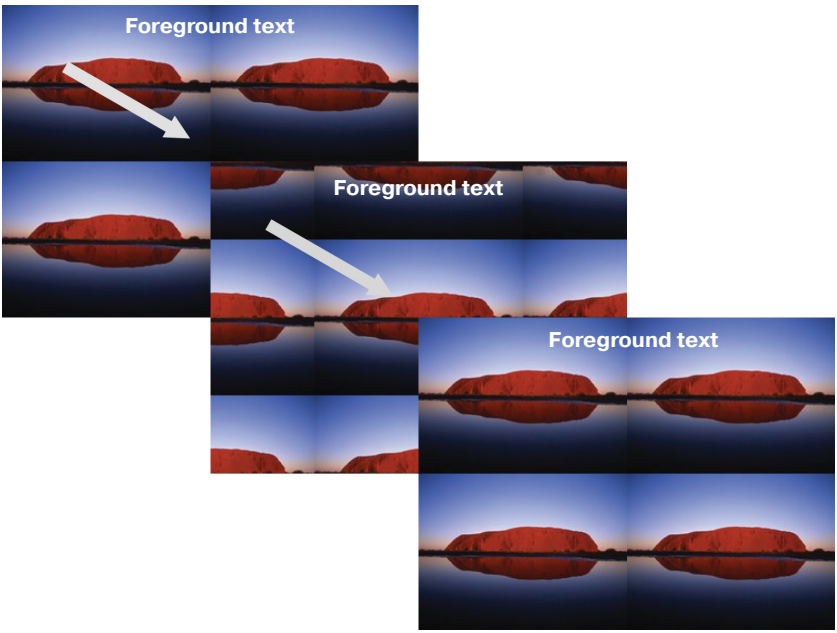


Рис. 11.2 ❖ Анимация позиции фонового изображения (в направлении стрелки) в моменты, соответствующие 0%, 50% и 100% от общей продолжительности эффекта

В дополнение к стандартным числовым значениям с необязательными единицами измерения позиция фонового изображения может определяться именованными значениями: `left`, `center` или `right` для смещений по горизонтали и `top`, `center` или `bottom` для смещений по

вертикали. Эти имена соответствуют числовым значениям 0%, 50% и 100% в каждом направлении. Необходимо также предусмотреть возможность указывать относительные смещения, начинающиеся с -= и +=.

```
.uluru { background-image: url(img/uluru.jpg);
    background-position: 'left top'; }
```

После подключения собственного обработчика анимации можно будет реализовать прокрутку фонового изображения, как показано ниже:

```
$('div.uluru').animate({'background-position': '200px 150px'});
```

> Для свойств с именами, содержащими дефис (-), допускается указывать имя в «верблюжьей» нотации, чтобы избавиться от кавычек. jQuery автоматически выполняет преобразование между этими двумя формами записи: `$('div.uluru').animate({backgroundPosition: '200px 150px'});`.

Прежде всего необходимо определить обработчик анимации для данного свойства и функцию для извлечения текущего значения свойства.

11.2.2. Объявление обработчика и извлечение значения свойства

Создание собственного расширения для механизма анимации следует начинать с определения функции чтения, извлекающей текущее значение свойства. В листинге 11.2 представлена заготовка объявления такого расширения.

Листинг 11.2. Определение расширения для jQuery 1.8.x

```
// ❶ Объявление анонимной функции
(function($) { // Скрывает из видимости внешнего кода,
               // устраняет конфликты с идентификатором $

// ❷ Добавить собственный обработчик анимации для требуемого свойства
$.Tween.propHooks['backgroundPosition'] = {
    // ❸ Определение функции чтения значения свойства
    get: function(tween) {
        return parseBackgroundPosition($(tween.elem).css(tween.prop));
    },
    // ❹ Определение функции записи значения в свойство
    set: setBackgroundPosition
};

// ❺ Немедленный вызов функции-обертки
})(jQuery);
```

Как и при создании любых других расширений, сначала определяется анонимная функция-обертка ❶, скрывающая код расширения от внешнего мира и гарантирующая возможность использования идентификатора $ для ссылки на объект jQuery за счет передачи его в виде параметра ❺.

Далее определяется обработчик анимации в виде объекта, подключаемого к точке интеграции $.Tween.propHooks, с именем, соответствующим имени свойства ❷. Если имя свойства содержит дефис (-), его можно заменить соответствующим именем в «верблюжьей» нотации, как показано здесь: backgroundPosition вместо background-position. Библиотека jQuery автоматически выполнит все необходимые преобразования имени с дефисом перед его использованием.

Объект обработчика анимации должен содержать две функции: функцию чтения для извлечения текущего значения свойства ❸ и функцию записи для сохранения нового значения в этом свойстве ❹. Вся остальная обработка выполняется внутренними функциями библиотеки. Обеим функциям в качестве аргумента (tween) передается объект, содержащий настройки анимации для данного свойства. В число его атрибутов входят: ссылка на текущий элемент DOM (elem), имя свойства (prop), используемая функция управления переходом (easing) и дополнительные параметры настройки, переданные в вызов функции animate (options).

В листинге11.3 демонстрируется реализация функции, извлекающей значения смещений из свойства.

Листинг 11.3. Извлечение значения из свойства background-position

```
/* Выполняет парсинг свойства background-position:
   смещения по горизонтали [вертикали]
   @param  value  (string) значение
   @return ([2][string, number, string]) извлеченные значения -
           признак относительного значения, величина, единицы измерения */
// ❶ Функция, интерпретирующая значение местоположения
function parseBackgroundPosition(value) {
    // ❷ Выделить горизонтальный и вертикальный компоненты
    var bgPos = (value || '').split(/ /);
    var presets = {center: '50%', left: '0%', right: '100%',
        top: '0%', bottom: '100%'};
    // ❸ Разбить значения смещений на величины и единицы измерения
    var decodePos = function(index) {
        var pos = (presets[bgPos[index]] || bgPos[index] || '50%').
            match(/^([+-]=)?([+-]?\d+(\.\d*)?)(.*)$/);
        bgPos[index] = [pos[1], parseFloat(pos[2]), pos[4] || 'px'];
    };
    // ❹ Обработать случай изменения только вертикальной позиции
```

```
    if (bgPos.length == 1 &&
            $.inArray(bgPos[0], ['top', 'bottom']) > -1) {
        bgPos[1] = bgPos[0];
        bgPos[0] = '50%';
    }
    // ❺ Декодировать значения смещений и вернуть их
    decodePos(0);
    decodePos(1);
    return bgPos;
}
```

Значением свойства является не простое число, поэтому необходимо найти в нем отдельные части, пригодные для дальнейшего использования. Для интерпретации таких значений определяется отдельная функция ❶. Так как значение свойства может состоять из двух частей – смещения по горизонтали и вертикали, прежде всего следует разделить их с помощью функции `split` ❷.

Каждая часть затем декодируется отдельно, и из нее извлекаются элементы значения позиции с помощью внутренней функции (`decodePos`) ❸. При необходимости именованные значения позиций преобразуются (с помощью объекта `presets`) в числовые или получают значение `50%` (`center`), если вообще не были указаны. Далее с применением регулярного выражения выполняется поиск операторов `+=` и `-=`, служащих признаками относительных значений, за которыми следуют числа (включая необязательный унарный минус и дробную часть), а также упоминание о единицах измерения. Все эти составляющие сохраняются в группах регулярного выражения (соответствующие им подвыражения заключены в круглые скобки `()`). Наконец, извлеченные части (признак относительного значения, величина в числовом выражении и единицы измерения) сохраняются в соответствующем элементе массива (`bgPos`). Такое разделение элементов упрощает дальнейшее их использование в процессе анимации.

Свойство `background-position` может содержать единственное значение, которое в этом случае интерпретируется как смещение по горизонтали и вертикали одновременно, если явно не указано, что значение является смещением по вертикали (`top` или `bottom`). В последнем случае в качестве значения смещения по горизонтали принимается значение `center`. Эту ситуацию следует проверить обязательно и интерпретировать значения соответственно ❹.

Затем к каждой части значения позиции поочередно применяются функция декодирования ❺ и обновленное значение, теперь содержащее извлеченные компоненты позиции, возвращается вызывающей программе.

Завершается создание обработчика анимации определением функции записи, изменяющей значение свойства.

11.2.3. Изменение значения свойства

Следующий шаг – изменение значения позиции в ходе анимации и сохранение этого значения обратно в свойстве элемента. В листинге 11.4 представлена реализация функции записи в объекте-обработчике анимации для свойства `background-position`.

Листинг 11.4. Установка значения в свойство background-position

```
/* Устанавливает значение для функции step.
   @param  tween  (object)  параметры анимации */
// ❶ Определение функции записи
function setBackgroundPosition(tween) {
    // ❷ Инициализировать настройки, если это еще не сделано
    if (!tween.set) {
        initBackgroundPosition(tween);
    }
    // ❸ Установить новое значение свойства
    $(tween.elem).css('background-position',
        ((tween.pos * (tween.end[0][1] - tween.start[0][1]) +
        tween.start[0][1]) + tween.end[0][2]) + ' ' +
        ((tween.pos * (tween.end[1][1] - tween.start[1][1]) +
        tween.start[1][1]) + tween.end[1][2]));
}
```

Для записи нового значения в свойство определяется внутренняя функция ❶. В качестве аргумента этой функции передается объект, содержащий информацию о текущем анимационном эффекте. Библиотека jQuery вызывает эту функцию неоднократно – на каждом шаге воспроизведения анимационного эффекта, поэтому старайтесь минимизировать объем работы, выполняемой ею.

Когда функция вызывается в первый раз, необходимо выполнить инициализацию: вычислить значения, которые не будут изменяться на всем протяжении анимационного эффекта. Чтобы избежать лишних затрат на инициализацию в каждом вызове, вход в эту процедуру контролируется флагом, устанавливаемым в объекте с параметрами настройки ❷. Функция инициализации представлена в листинге 11.5.

На каждом шаге анимации требуется изменить значение свойства текущего элемента (`tween.elem`), сохранив в нем новое значение ❸ вызовом стандартной функции `css`. Новое значение вычисляется на основе значения доли истекшего времени анимации (`tween.pos`), определяемого самой библиотекой jQuery, которое равно 0 в начале

анимации и 1 – в конце. В общем случае достаточно просто умножить значение доли истекшего времени на разность между конечным и начальным значениями свойства, сложить результат с начальным значением и добавить в конец спецификатор единиц измерения, как показано ниже:

```
(tween.pos * (tween.end - tween.start) + tween.start) + tween.unit
```

В случае со свойством `background-position` описанные действия следует выполнить для каждого смещения – по горизонтали и вертикали – перед их объединением в новое составное значение.

Листинг 11.5. Инициализация обработчика анимации для свойства background-position

```
/* Инициализация анимационного эффекта.
   @param  tween  (object) параметры анимации */
// ❶ Определение функции инициализации анимации
function initBackgroundPosition(tween) {
    // ❷ Декодировать начальные позиции
    tween.start = parseBackgroundPosition(
        $(tween.elem).css('backgroundPosition'));
    // ❸ Декодировать конечные позиции
    tween.end = parseBackgroundPosition(tween.end);
    // ❹ Для каждого декодированного значения...
    for (var i = 0; i < tween.end.length; i++) {
        // ❺ ...вычислить относительные позиции
        if (tween.end[i][0]) { // Относительная позиция
            tween.end[i][1] = tween.start[i][1] +
                (tween.end[i][0] == '-=' ? -1 : +1) * tween.end[i][1];
        }
    }
    // ❻ Установить флаг, указывающий, что инициализация выполнена
    tween.set = true;
}
```

Чтобы уменьшить объем работы, выполняемой на каждом шаге анимации, здесь определяется функция инициализации ❶, которая вызывается один раз, в самом начале процесса. Конкретная реализация этой функции зависит от структуры свойства, участвующего в анимации, но если говорить в целом – она извлекает составные части начального и конечного значений, чтобы их проще было объединить в функции записи, представленной выше.

Для анимации свойства `background-position` необходимо вычислить начальное и конечное его значения и сохранить в атрибутах объекта анимации (`tween`) в виде массивов составных частей значений, полученных с помощью функции `parseBackgroundPosition` (❷ и ❸).

Для каждого конечного значения ❹ проверяется, не является ли оно относительным, то есть не начинается ли с оператора += или -=, путем проверки первого элемента соответствующего массива tween.end ❺. Если значение является относительным, оно прибавляется к начальному (tween.start) или вычитается из него, чтобы получить конечное значение.

В заключение устанавливается флаг, указывающий, что инициализация выполнена ❻ и ее не требуется повторять для данного анимационного эффекта с данным элементом.

Теперь новое расширение Background Position готово к использованию, и с его помощью можно реализовать анимационный эффект, основанный на изменении позиции фонового изображения, по аналогии с любыми другими свойствами элемента.

Как отмечалось выше, в версии jQuery 1.8 порядок выполнения анимационных эффектов изменился, поэтому в следующем разделе описывается, как реализовать тот же эффект в более старых версиях библиотеки.

11.2.4. Анимация свойства background-position в jQuery 1.7

При использовании версии библиотеки jQuery ниже 1.8.0 собственный обработчик анимации необходимо добавлять в свойство $.fx.step в виде ссылки на функцию записи. К счастью, внутренний порядок выполнения этапов анимации почти не отличается в старых и новых версиях jQuery, поэтому есть возможность повторно использовать большую часть кода, созданного для новой версии, как показано в листинге 11.6.

Листинг 11.6. Анимация свойства background-position в jQuery 1.7

```
// ❶ Объявление анонимной функции
 (function($) { // Скрывает из видимости внешнего кода,
                // устраняет конфликты с идентификатором $

// ❷ Подключить собственный обработчик анимации
//    для атрибута background-position
$.fx.step['backgroundPosition'] = setBackgroundPosition;

// ❸ Немедленный вызов функции-обертки
})(jQuery);
```

Как и в новых версиях jQuery, определение расширения заключается в вызове анонимной функции ❶, скрывающей код расшире-

ния и гарантирующей возможность использования идентификатора $ для ссылки на объект jQuery ❸. Внутри этой функции выполняется подключение функции записи нового значения к точке интеграции $.fx.step ❷. Напомню еще раз, если имя свойства содержит дефис (-), при определении функции step его можно заменить соответствующим именем в «верблюжьей» нотации. Функция setBackgroundPosition осталась той же, что и в реализации для более новой версии jQuery.

Предусмотрев проверку версии инфраструктуры поддержки анимации в jQuery, можно выбирать, каким способом определять обработчик анимации.

Листинг 11.7. Выбор способа определения обработчика анимации

```
var usesTween = !!$.Tween;

// ❶ Определить версию инфраструктуры поддержки анимации
if (usesTween) { // jQuery 1.8+
    // ❷ Определить обработчик для jQuery 1.8+
    $.Tween.propHooks['backgroundPosition'] = {
        get: function(tween) {
            return parseBackgroundPosition(
                $(tween.elem).css(tween.prop));
        },
        set: setBackgroundPosition
    };
}
else { // jQuery 1.7-
    // ❸ Определить обработчик для jQuery 1.7-
    // Подключить собственный обработчик анимации
    // для атрибута background-position
    $.fx.step['backgroundPosition'] = setBackgroundPosition;
};
```

Действие конструкции !! объясняется в разделе 3.2.1.

Определить текущую версию инфраструктуры поддержки анимации можно, проверив наличие свойства $.Tween ❶. По результатам проверки либо создается новый элемент в объекте $.Tween.propHooks ❷ (новая версия инфраструктуры), либо добавляется функция $.fx.step ❸ (старая версия инфраструктуры). Теперь расширение анимации будет работать с любыми версиями jQuery.

В версиях jQuery ниже 1.5 стандартный механизм анимации повреждает начальное значение параметра background-position, поэтому код, представленный здесь, не будет работать в этих ранних версиях.

11.2.5. Законченное расширение

Итак, мы закончили разработку расширения jQuery Background Position. С его помощью можно реализовать анимационный эффект, выражающийся в перемещении фона элемента. Данное расширение также позволяет использовать все возможности инфраструктуры поддержки анимации в библиотеке jQuery, такие как позиционирование с применением единиц измерения `em` или процентов, осуществлять относительные смещения, выбирать функцию управления переходом и передавать собственную функцию для вызова по окончании воспроизведения анимационного эффекта. Полный код расширения доступен для загрузки на сайте книги.

Это нужно знать

Стандартная реализация анимации в библиотеке jQuery может манипулировать только простыми числовыми значениями.
Для анимации свойств со сложными значениями создавайте собственные обработчики анимации.
Обработчики анимации должны подключаться к точке интеграции `$.Tween.propHooks`.
Для чтения и записи значений свойств предоставляйте специализированные функции чтения и записи.
В версиях jQuery ниже 1.8 следует использовать точку интеграции `$.fx.step`.

Попробуйте сами

Реализуйте обработчик анимации для свойства `border-width`. Значение этого свойства может состоять из четырех простых чисел, определяющих толщину верхней, правой, нижней и левой сторон рамки соответственно. Если в свойстве содержится только одно значение, оно применяется ко всем сторонам. Если в свойстве содержатся два значения, они применяются к верхней/нижней и правой/левой сторонам. Если в свойстве содержатся три значения, отсутствующее значение для левой стороны рамки принимается равным значению для правой стороны.

11.3. В заключение

Библиотека jQuery поддерживает возможность анимации различных свойств элементов, вызывая тем самым разнообразные визуальные эффекты в веб-странице. Помимо встроенных функций `show`, `hide`, `fade` и `slide`, для воспроизведения анимационных эффектов можно использовать функцию `animate`. Однако автоматически механизмом анимации поддерживаются только свойства с простыми числовыми значениями и единицами измерения.

Для анимации свойств, значения которых не соответствуют стандартному формату, необходимо загрузить или создать свое расширение анимации, способное правильно интерпретировать такие значения. Библиотека jQuery UI включает расширения для анимации свойств, содержащих значения цвета, которые позволяют манипулировать ими наравне со стандартными числовыми значениями.

На примере реализации обработчика анимации для свойства стиля `background-position` мы увидели, как создавать свои расширения, которые смогут извлекать необходимую информацию из значений свойств, вычислять текущие значения в ходе анимации и сохранять их в элементах. Такие расширения дают возможность воспроизводить анимационные эффекты на основе неподдерживаемых свойств наряду со стандартными.

В следующей главе мы исследуем инфраструктуру поддержки технологии Ajax в библиотеке jQuery и узнаем, как расширять ее, с целью удовлетворения дополнительных потребностей.

Глава 12

Расширение поддержки Ajax

Эта глава охватывает следующие темы:

- инфраструктура поддержки технологии Ajax в библиотеке jQuery;
- добавление предварительных фильтров Ajax;
- добавление транспортов Ajax;
- добавление преобразователей Ajax.

Поддержка технологии *Ajax* (Asynchronous JavaScript and XML – асинхронный JavaScript и XML) – это одна из ключевых особенностей библиотеки jQuery. Она упрощает отправку запросов серверу, получение содержимого, его обработку и обновление информации на странице, устраняя тем самым необходимость полностью обновлять страницу. Для этого достаточно указать адрес URL, параметры запроса, а также функцию для обработки данных, которая будет вызвана при получении ответа от сервера, как показано ниже:

```
$.ajax('product.php', {data: {prod_id: 'AB1234'},
success: function(info) {...}});
```

В библиотеке jQuery имеется также несколько вспомогательных функций, инкапсулирующих доступ к механизмам Ajax. Для отправки простого запроса и получения ответа можно использовать функцию `get`, а для передачи запроса с большим количеством параметров – функцию `post`. Для загрузки данных определенного типа можно пользоваться функциями `getScript` и `getJSON`, которые запрашивают у сервера получение сценария JavaScript или данных в формате *JSON* (JavaScript Object Notation – форма записи объектов JavaScript) соответственно. Чтобы запросить фрагмент разметки HTML для вставки непосредственно в элемент страницы, можно воспользоваться функцией `load` этого элемента.

```
$.get('product.php', {prod_id: 'AB1234'}, function(info) {...});
$.getScript('product.js');
$('#mydiv').load('product.php', {prod_id: 'AB1234'}, function(info) {...});
```

Дополнительно библиотека jQuery предоставляет возможность определить значения по умолчанию параметров для всех запросов Ajax и зарегистрировать обработчики событий, возникающих в течение жизненного цикла механизма Ajax.

Несмотря на богатство встроенных функциональных возможностей, чтобы получить данные в необычном формате, вам может потребоваться самостоятельно реализовать обработку получаемого содержимого. К счастью, в инфраструктуре поддержки технологии Ajax имеется несколько точек интеграции, используя которые, можно реализовать собственные процедуры загрузки и обработки содержимого.

12.1. Инфраструктура поддержки Ajax

Поддержка технологии Ajax в библиотеке jQuery основана на использовании объекта `XMLHttpRequest` (или соответствующего объекта ActiveX в некоторых версиях IE) для выполнения фактической загрузки удаленного содержимого. Каждый запрос Ajax выполняется библиотекой jQuery в несколько этапов. Вы можете внедряться в каждый из этих этапов и предусматривать выполнение своих операций в зависимости от требований.

Выполнение запроса начинается с применения предварительных фильтров, которые могут оказывать влияние на дальнейшее течение запроса, перед выбором транспортного механизма для фактической загрузки данных. После получения содержимого оно может преобразовываться в различные форматы в зависимости от требований пользователя. На рис. 12.1 изображена последовательность операций, выполняемых в процессе простого запроса Ajax.

Управление процессом в целом осуществляется выбором типа запрашиваемых данных. В число стандартных поддерживаемых типов входят `text`, `html`, `xml`, `script`, `json` и `jsonp`. Если тип содержимого не указан явно, библиотека jQuery попытается определить его по возвращаемому типу MIME. Если тип возвращаемых данных не соответствует запрошенному, запускается процедура преобразования.

Начиная с версии 1.5 библиотека обертывает встроенный в браузеры объект `XMLHttpRequest` дополнительными функциональными возможностями, реализованными в виде объекта `jqXHR`. Этот объект

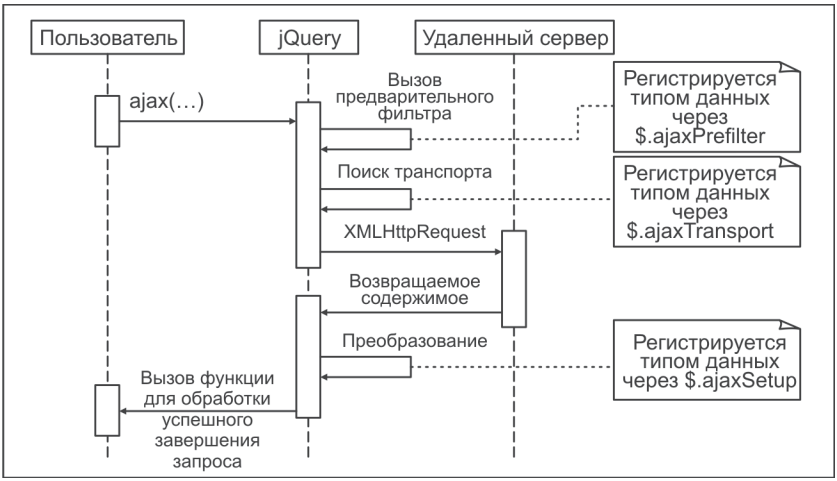


Рис. 12.1 ❖ Диаграмма стандартной процедуры выполнения запроса Ajax с точками интеграции

также действует как объект `Deferred`[1], что дает возможность регистрировать дополнительные функции обратного вызова для обработки успешного или неудачного завершения запроса Ajax.

> Функции `success`, `error` и `complete` объекта `jqXHR` не рекомендуются для использования, начиная с версии jQuery 1.8. Вместо них следует применять соответствующие им функции `done`, `fail` и `always`.

12.1.1. Предварительные фильтры

Предварительный фильтр – это функция, вызываемая перед фактической отправкой запроса серверу. Определив свою функцию, с ее помощью можно выполнить предварительную обработку запроса и изменить порядок дальнейшей его обработки, что может пригодиться для создания собственных типов данных. Например, с помощью предварительного фильтра можно добавить в запрос дополнительные заголовки и даже отменить его выполнение.

Предварительные фильтры вызываются после сериализации параметров – после преобразования параметра `data` в строку (предполагается, что параметр `processData` имеет значение `true`), но перед тем, как

---

[1] См. раздел «Category: Deferred Object» (http://api.jquery.com/category/deferred-object/) в документации с описанием jQuery API (на русском языке: http://slyweb.ru/jquerymain/jquery.php?razdel=Deferred. – *Прим. перев.*).

инфраструктура поддержки Ajax приступит к поиску соответствующего транспорта. Выбор предварительного фильтра определяется типом данных, таким как `html` или `script`. Имеется возможность организовать вызов предварительного фильтра для данных любых типов, если при регистрации в качестве типа указать шаблонный символ `*`. В первую очередь вызываются фильтры, специализированные для данного типа, а затем применяются универсальные фильтры.

Новый предварительный фильтр можно зарегистрировать вызовом функции `$.ajaxPrefilter`, передав ей тип данных, который должен обрабатываться фильтром, и саму функцию, реализующую фильтр. В качестве аргументов функция фильтра получит набор параметров настройки Ajax и ссылку на объект `jqXHR`, с помощью которого будет выполнен запрос. Чтобы отменить запрос, следует вызвать метод `abort` этого объекта.

Например, чтобы изменить строку идентификации агента для всех запросов с типом данных `html`, можно использовать следующий предварительный фильтр:

```
$.ajaxPrefilter('html', function(options, originalOptions, jqXHR) {
    jqXHR.setRequestHeader('User-Agent', 'Unknown');
});
```

12.1.2. Транспорт

Транспорт – это механизм извлечения данных со стороны сервера. Обычно в качестве транспорта используется объект `XMLHttpRequest`, но иногда для загрузки содержимого предпочтительнее использовать другие средства, такие как атрибут `src` элемента `img` (чтобы загрузить изображение).

Новые транспорты регистрируются вызовом функции `$.ajaxTransport`, которой передаются: тип данных, извлекаемых с применением данного транспорта, и функция, возвращающая объект, который реализует доставку данных. Объект транспорта должен предоставлять две функции обратного вызова: одна должна выполнять фактическое извлечение данных, а другая должна выполнять заключительные операции в случае отмены запроса. Как и в случае с предварительными фильтрами, имеется возможность зарегистрировать транспорт для типа данных `*`. Если в странице будет определено несколько транспортов, соответствующих одному и тому же типу данных, библиотека вызовет только первый встретившийся, при этом предпочтение будет отдано более специализированному транспорту.

Например, чтобы предотвратить доступ к любым документам XML, можно использовать следующий транспорт:

```
$.ajaxTransport('xml', function(options, originalOptions, jqXHR) {
    return {
        send: function(headers, complete) {
            complete('403', 'Forbidden', {});
        },

        abort: function() {}
    };

});
```

Сама библиотека jQuery использует эту возможность для выполнения междоменных запросов на получение данных с типом `script`.

12.1.3. Преобразователи

Данные не всегда поступают со стороны сервера в требуемом формате, пригодном для немедленного использования. Поэтому иногда возникает необходимость в *преобразователях* – функциях, принимающих данные в виде текста и возвращающих их в желаемом формате.

Например, при запросе XML-документа можно указать в параметре `dataType` тип данных `xml`. В результате текст XML-документа, полученный от сервера, будет преобразован в дерево XML DOM. Благодаря этому можно будет немедленно начать обход дерева DOM и извлечь информацию, соответствующую текущим потребностям. Преобразователи регистрируются в атрибуте `converters` объекта, передаваемого функции `$.ajaxSetup`. При регистрации необходимо в виде ключа, который будет применяться для идентификации преобразователя, указать исходный и конечный форматы (в виде одной строки), а в виде значения передать ссылку на функцию, выполняющую преобразование. Чтобы показать, что преобразователь может применяться к данным любого типа, в соответствующей позиции ключа можно использовать символ `*`. Например, упомянутый выше преобразователь текстовых данных в тип `xml` можно определить, как показано ниже (он выполняет преобразование с помощью встроенной функции `parseXML` из библиотеки jQuery):

```
ajaxSettings: {
    ...
    converters: {
        ...
        // Преобразует текст в данные типа xml
        "text xml": jQuery.parseXML
```

```
    },
    ...
}
```

Точно так же можно определить собственные преобразователи для представления данных в альтернативных форматах.

Все эти точки интеграции подробно описываются в следующих разделах.

12.2. Добавление предварительного фильтра

Предварительные фильтры дают возможность обработать запрос Ajax перед отправкой и, возможно, изменить порядок его обработки. Например, увеличить значение параметра настройки `timeout` при работе с медленным сервером или даже вообще отменить отправку запроса. Сама библиотека jQuery использует предварительные фильтры для обработки данных типа `json` и `jsonp`, позволяя устанавливать соответствующие функции обратного вызова для таких запросов. Далее в этом разделе вы увидите два примера реализации предварительных фильтров: один изменяет тип данных в запросе, а другой вообще отменяет отправку запроса.

12.2.1. Изменение типа данных

Предварительные фильтры дают возможность изменять настройки объекта `XMLHttpRequest` (заключенного в объект `jqXHR`) перед отправкой запроса. Они могут также использоваться для изменения типа данных в запросе, чтобы в результате запроса были возвращены данные желаемого типа. В результате этого все последующие операции будут интерпретировать данные как данные нового типа.

В листинге 12.1 показано, как можно изменить тип данных с помощью предварительного фильтра.

Листинг 12.1. Изменение типа данных

```
/* Установить тип данных CSV. */
// ❶ Определить предварительный фильтр для любых типов данных
$.ajaxPrefilter(function(options, originalOptions, jqXHR) {
    // ❷ Если запрошен файл *.csv...
    if (options.url.match(/.*\.csv/)) {
        // ❸ ...изменить тип данных
        return 'csv';
    }
});
```

Для регистрации функции нового предварительного фильтра используется функция `$.ajaxPrefilter` ❶. Обратите внимание, что в данном случае не требуется указывать тип данных, для которого устанавливается фильтр, потому что тип `*` («любой») подразумевается по умолчанию. Нам требуется, чтобы при запросе любого файла с расширением `.csv` его содержимое интерпретировалось бы как данные типа `csv`, поэтому мы проверяем указанный адрес URL на присутствие в нем расширения `.csv` ❷ и, если условие выполняется, возвращаем новый тип данных ❸.

Чтобы продолжить работу, не меняя типа данных, ничего возвращать не нужно. В разделе 12.4 описывается, как преобразовать текстовые данные в формат CSV с помощью преобразователя.

12.2.2. Отмена запроса Ajax

Иногда случается так, что бывает желательно вообще запретить выполнение всех запросов Ajax или, по крайней мере, некоторых из них. В листинге 12.2 показано, как с помощью предварительного фильтра реализовать отмену некоторых запросов.

Листинг 12.2. Отмена выполнения запросов Ajax

```
/* Отмена выполения запросов Ajax. */
// ❶ Список типов данных, которые запрещено запрашивать
$.ajax.disableDataTypes = [];

// ❷ Определение предварительного фильтра для всех типов данных
$.ajaxPrefilter('*', function(options, originalOptions, jqXHR) {
    // ❸ Запретить выполнение запросов для данных указанных типов
    if ($.inArray(options.dataType, $.ajax.disableDataTypes) > -1) {
        jqXHR.abort();
    }
});
```

Здесь сначала определяется список запрещенных типов данных ❶. Пользователь может просто добавлять в этот массив выбранные им типы по мере необходимости:

```
$.ajax.disableDataTypes.push('html');
```

Затем вызовом функции `$.ajaxPrefilter` создается предварительный фильтр ❷. Ей передается тип данных фильтра и функция, реализующая его. Тип данных определяется как строка с названиями одного или нескольких типов данных, разделенных пробелами, такими как `html` или `json jsonp`. В качестве типа данных можно также передать шаблонный символ `*` (или вообще опустить аргумент с типом

данных), чтобы разрешить применение предварительного фильтра ко всем типам данных. Кроме того, можно указать, что фильтр должен вызываться перед любыми другими фильтрами, добавив знак + перед названием типа.

Функция предварительного фильтра принимает несколько аргументов: `options` хранит все параметры настройки механизма Ajax для данного запроса, независимо от того, имеют они значение по умолчанию или были определены в вызове одной из функций отправки запроса Ajax, в аргументе `originalOptions` передаются только настройки, явно указанные пользователем, а в аргументе `jqXHR` – ссылка на объект `jqXHR`. Функция фильтра может анализировать параметры настройки (измененные пользователем или имеющие значения по умолчанию) и изменять запрос, выполняя операции с объектом `jqXHR`. В данном случае функция проверяет присутствие типа данных запроса в списке запрещенных и прерывает выполнение запроса, если проверка дает положительный результат ❸.

В число других операций, которые можно выполнять с объектом `jqXHR`, входят также добавление заголовков в запрос вызовом функции `setRequestHeader(name, value)` или изменение MIME-типа вызовом функции `overrideMimeType(mimeType)`.

Чтобы удалить название типа данных из списка запрещенных, можно воспользоваться функцией `$.map` из библиотеки jQuery. Например, удалить название типа данных `html` из списка можно так:

```
$.ajax.disableDataTypes = $.map($.ajax.disableDataTypes, function(v) {
    return (v == 'html' ? null : v);
});
```

Этот фильтр можно дополнить анализом типа запроса (`GET` или `POST`) и других параметров настройки механизма Ajax.

12.3. Добавление транспорта Ajax

Транспорты Ajax обеспечивают механизм загрузки запрошенного содержимого. Роль транспорта по умолчанию играет стандартный объект `XMLHttpRequest`, однако есть возможность определить альтернативные средства доставки данных того или иного типа, добавив собственную функцию, реализующую транспортировку. Поддерживается также возможность изменить процедуру получения данных стандартных типов и добавить в нее собственную функциональность. Обе эти разновидности расширений иллюстрируются примерами ниже.

12.3.1. Загрузка изображений

Предположим, что требуется предварительно загрузить изображения для страницы. Для этого можно было бы создать элементы `Image`, настроить их соответственно, чтобы инициировать загрузку изображений, и вызвать свою функцию, когда загрузка закончится. Но можно также задействовать инфраструктуру поддержки Ajax и получить в свое распоряжение все ее преимущества, такие как аутентификация и обработка ошибок.

Чтобы загрузить изображения с применением технологии Ajax, можно определить функцию транспорта для данных типа `image`, которая знает, как обрабатывать данные в этом формате, и использует возможности объекта `Image` для выполнения фактической загрузки. В листинге 12.3 демонстрируется определение функции-транспорта, решающей эту задачу.

Листинг 12.3. Загрузка изображения

```
/* ❶ Определение транспорта для изображений. */
$.ajaxTransport('image', function(options, originalOptions, jqXHR) {
    // ❷ Только для асинхронных запросов GET
    if (options.type === 'GET' && options.async) {
        var image;
        // ❸ Вернуть объект транспорта
        return {
            // ❹ Определение функции отправки запроса
            send: function(headers, complete) {
                image = new Image();
                // ❺ Функция, вызываемая по завершении запроса
                function done(status) {
                    if (image) {
                        var statusText = (status == 200 ?
                            'success' : 'error');
                        var tmp = image;
                        image = image.onreadystatechange =
                            image.onerror = image.onload = null;
                        // ❻ Вызвать функцию complete
                        complete(status, statusText, {image: tmp});
                    }
                }
                // ❼ Инициализировать функции обратного вызова
                image.onreadystatechange = image.onload = function() {
                    done(200);
                };
                image.onerror = function() {
                    done(404);
                };
```

```
                // ❽ Загрузить изображение
                image.src = options.url;
            },

            // ❾ Обработать ситуацию отмены запроса
            abort: function() {
                if (image) {
                    image = image.onreadystatechange =
                        image.onerror = image.onload = null;
                }
            }
        };
    }
});
```

Определение транспорта для загрузки данных типа `image` производится вызовом функции `$.ajaxTransport` ❶. Как и в случае с предварительными фильтрами, в первом аргументе можно передать сразу несколько типов данных, разделив их пробелами, или указать шаблонный символ `*`, чтобы определить транспорт для всех типов данных, а также добавить префикс `+` перед названием типа, чтобы переместить данный транспорт на первое место в списке транспортов для данного типа. В аргументе `options` функции транспорта передается полный комплект параметров настройки для данного запроса, включая значения по умолчанию; в аргументе `originalOptions` передаются только настройки, явно указанные пользователем. В аргументе `jqXHR` передается ссылка на объект `jqXHR`, обычно используемый для выполнения запросов. Так как в данном случае загрузка изображений осуществляется альтернативным механизмом, последний аргумент просто игнорируется.

Новый транспорт применяется только к асинхронным запросам типа `GET` (из-за ограничений фактического используемого механизма), поэтому необходимо гарантировать соблюдение этого условия ❷. Если транспорт может быть применен к данному запросу, возвращается объект транспорта, позволяющий библиотеке jQuery запустить процесс загрузки в соответствующий момент времени ❸. Объект транспорта содержит две функции: `send`, инициирующую загрузку, и `abort`, выполняющую заключительные операции в случае отмены запроса или появления ошибки в процессе его выполнения.

Функция `send` ❹ используется вместо стандартного механизма Ajax обработки запроса для указанного типа данных. В качестве аргументов она принимает ссылку на заголовки запроса (`headers`) и функцию для вызова по окончании выполнения запроса (`complete`).

Так как для загрузки данных предполагается использовать возможности стандартного элемента Image, необходимо сначала создать новый элемент Image.

Далее определяется функция для вызова по окончании загрузки изображения ❺. Внутри этой функции определяются обработчики успешного и неудачного завершения загрузки, которые устанавливают соответствующий код состояния. Затем выполняется очистка созданного элемента Image сбросом ссылок на функции обратного вызова и самой переменной в значение null, что обеспечивает утилизацию памяти сборщиком мусора и позволяет избежать утечек памяти. Изображение все еще остается доступным через локальную переменную tmp, но оно недоступно за пределами функции done.

В заключение вызывается функция complete, переданная в аргументе функции send, чтобы сообщить инфраструктуре поддержки Ajax результаты выполнения запроса ❻. В аргументах ей передаются: числовая и текстовая версии кода состояния, объект с подробной информацией об ответе и строка (необязательно) со всеми заголовками ответа – по одному в строке. Этот объект должен содержать атрибут с именем, совпадающим с названием типа данных (image в данном случае), ссылающийся на фактический результат. В данном случае возвращается ссылка на элемент Image.

После определения функции, выполняющей загрузку, производится определение функций для обработки стандартных событий элемента Image, передающих соответствующие коды состояния ❼. Последний шаг – запуск процесса загрузки записью адреса URL в атрибут src элемента Image ❽, по окончании которого будет вызван один из зарегистрированных обработчиков.

Функция abort в возвращаемом объекте транспорта ❾ дает возможность выполнить заключительные операции в случае отмены или неудачного завершения запроса. В данном случае вновь выполняется очистка внутренних переменных элемента Image путем присваивания им значения null.

Чтобы воспользоваться этим альтернативным транспортом, можно выполнить запрос данных типа image и получить ссылку на загруженное изображение в виде аргумента функции success.

```
$.ajax({url: 'img/uluru.jpg', dataType: 'image', success: function(image) {
    $('#img1').replaceWith(image);
}});
```

Реализация этого транспорта иллюстрирует, как можно использовать альтернативные механизмы загрузки для получения данных

со стороны сервера. Следующий пример демонстрирует, как можно имитировать обычную процедуру загрузки разметки HTML, переопределив стандартный транспорт.

12.3.2. Имитация загрузки HTML для нужд тестирования

При тестировании расширения, использующего технологию Ajax, может оказаться желательным избежать попыток обращения за содержимым к действующему сайту, чтобы не попасть в зависимость от наличия соединения. С этой целью можно переопределить транспорт по умолчанию, подставив свой, возвращающий предопределенное содержимое. При этом тестовые данные могут храниться вместе с самими тестами, что снижает вероятность их потери.

Чтобы обеспечить полную имитацию удаленного доступа, необходимо реализовать отображение имен запрашиваемых файлов в тестовое содержимое. Кроме того, можно определить величину задержки перед возвратом содержимого, чтобы сымитировать сетевые задержки. Для расширения охвата тестируемых ситуаций можно также предусмотреть управление возвращаемым кодом состояния.

В листинге 12.4 демонстрируется, как можно определить транспорт Ajax, имитирующий выполнение запросов `GET` на получение данных типа `html`.

Листинг 12.4. Имитация загрузки HTML для нужд тестирования

```
/* Имитация загрузки HTML. */
// ❶ Определить отображение файлов
$.ajax.simulateHtml = {};

// ❷ Переопределить транспорт для данных типа html
$.ajaxTransport('html', function(options, originalOptions, jqXHR) {
    // ❸ Только для запросов типа GET
    if (options.type === 'GET') {
        var timer;
        // ❹ Вернуть объект транспорта
        return {
            // ❺ Определение функции отправки запроса
            send: function(headers, complete) {
                // ❻ Получить имя файла и настройки
                var fileName = options.url.replace(
                    /.*\/([^\/]+)$/, '$1');
                var simulate = $.ajax.simulateHtml[fileName] ||
```

```
                $.ajax.simulateHtml['default'];
            // ❼ Выполнить задержку
            timer = setTimeout(function() {
                // ❽ Вызвать функцию complete
                complete(simulate.html ? 200 : 404,
                    simulate.html ? 'success' : 'error',
                    {html: simulate.html});
            }, Math.random() * simulate.variation + simulate.delay);
        },

        // ❾ Обработать ситуацию отмены запроса
        abort: function() {
            clearTimeout(timer);
        }
    };
}
});
```

Прежде всего здесь объявляется объект (`$.ajax.simulateHtml`) для отображения страниц в возвращаемое содержимое ❶. Ключами атрибутов в этом объекте могут быть простые имена запрашиваемых файлов, при необходимости содержащие доменное имя сервера и путь. Значением каждого атрибута является объект с несколькими полями: `html` – хранит фактически возвращаемое содержимое, `delay` – минимальная задержка в миллисекундах перед возвратом содержимого, и `variation` – максимальная дополнительная задержка в миллисекундах (определяется как случайное значение для каждого отдельного запроса). Если содержимое определено как пустая строка, возвращается код состояния 404 (страница не найдена). Чтобы обеспечить обработку запросов с любыми именами файлов, следует включить элемент с индексом `default`. Например, ниже показано, как добавить отображение для файла `test.html`:

```
$.ajax.simulateHtml['default'] = {delay: 500, variation: 1000, html: ''};
$.ajax.simulateHtml['test.html'] = {delay: 500, variation: 1000,
    html: '<p>Try this instead</p>'};
```

Чтобы переопределить транспорт для данных типа `html`, используемый по умолчанию, нужно определить новый транспорт для этого типа ❷. Параметры функции транспорта остаются теми же, что и в предыдущем примере `image`: все параметры настройки, только указанные пользователем и ссылка на объект `jqXHR`. Альтернативный транспорт должен использоваться лишь для обработки запросов `GET`, поэтому, прежде чем продолжить, следует проверить выполнение этого условия ❸.

Функция транспорта возвращает объект транспорта, который библиотека jQuery сможет использовать для обработки запросов Ajax ❹. Функция `send` в объекте транспорта ❺ вызывается, когда производится запрос на получение содержимого указанного типа, и получает в виде аргументов заголовки запроса и функцию для вызова по окончании выполнения запроса.

Внутри этой функции сначала извлекается имя запрошенного файла ❻. Для этого используется регулярное выражение, совпадающее со всем (`.*`), вплоть до последнего слэша (`\/`), и захватывающее остаток без слэшей (`([^\/]+)`) до конца строки (`$`). Затем сохраненный остаток (доступный через `$1`) замещает совпавшую строку (всю), в результате чего остается только имя файла. Потом имя файла отображается в соответствующий объект с информацией об ответе, хранящийся в `$.ajax.simulateHtml`, или в объект `default` с параметрами ответа по умолчанию, если отображение для искомого файла не найдено.

Для имитации сетевых задержек используется стандартная функция `setTimeout` ❼, с помощью которой вносится постоянная задержка со случайной составляющей. По истечении времени задержки вызывается функция `complete`, переданная функции `send`, чтобы уведомить инфраструктуру поддержки Ajax о доступности содержимого ❽. Как и в предыдущем примере, функции `complete` передаются числовая и строковая версии кода состояния (если содержимое отсутствует, возвращается код 404 и соответствующий текст с сообщением об ошибке), объект с соответствующим содержимым в атрибуте, имя которого совпадает с именем типа данных (`html` в данном случае) и необязательная строка со всеми заголовками ответа.

Если по какой-то причине выполнение запроса было отменено, будет вызвана функция `abort` объекта транспорта, которая дает возможность выполнить заключительные операции ❾. В данном случае запрос можно отменить, прервав работу таймера, пока он запущен, вызовом `clearTimeout`.

Чтобы задействовать этот транспорт внутри теста QUnit (см. главу 7), необходимо определить отображение для имен файлов, как было показано выше на примере страницы `test.html`. После этого можно будет выполнить запрос Ajax с именем этой страницы и проверить соответствие полученного содержимого ожидаемому. Результат выполнения теста, представленного в листинге 12.5, изобра-

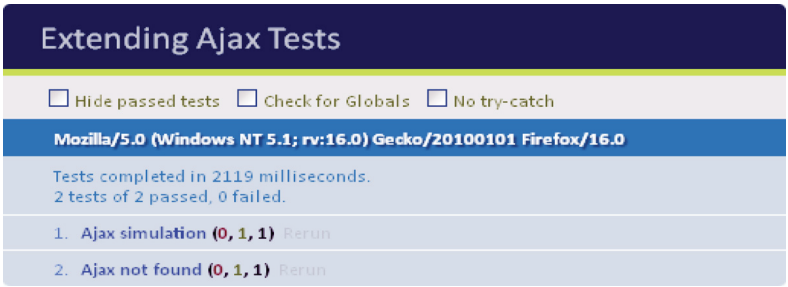


Рис. 12.2 ❖ Результат тестирования нового транспорта Ajax

жен на рис. 12.2. Не забывайте, что тест должен быть определен как асинхронный, потому что в работе участвует механизм поддержки Ajax.

Листинг 12.5. Тестирование транспорта, имитирующего загрузку HTML

```
// ❶ Определение асинхронного теста
asyncTest('Ajax simulation', function() {
    // ❷ Ожидается одна проверка
    expect(1);
    // ❸ Загрузить тестовую страницу с помощью механизма Ajax
    $.ajax('test.html', {dataType: 'html', success: function(data) {
        // ❹ Проверить загруженное содержимое
        equal($(data).text(), 'Try this instead', 'Ajax substitution');
        // ❺ Продолжить тестирование
        start();
    }});
});

// ❻ Проверить ошибку "страница не найдена"
asyncTest('Ajax not found', function() {
    expect(1);
    $.ajax('other.html', {dataType: 'html', success: function(data) {
        // ❼ Ошибка, если запрошенная страница найдена
        ok(false, 'Page found');
        start();
    }, error: function(jqXHR, textStatus, errorThrown) {
        // ❽ Проверить ошибку
        ok(jqXHR.status == 404 && textStatus == 'error', 'Page missing');
        // ❾ Продолжить тестирование
        start();
    }});
});
```

Вызовом asyncTest вместо test определяется асинхронный тест ❶ и указывается ожидаемое число проверок в этом тесте ❷. Далее вызывается функция ajax, выполняющая вызов для загрузки страницы test.html ❸. Этот вызов должен быть обработан подставным транспортом из листинга 12.4. Обратите внимание на необходимость определения типа данных dataType в вызовах Ajax, чтобы вынудить инфраструктуру использовать новую функциональность. Внутри функции success проверяется, совпадает ли полученное содержимое с данными, хранящимися в отображении ❹. Так как тест имеет асинхронную природу, необходимо вызвать функцию start, чтобы известить фреймворк тестирования QUnit, что тестирование завершено и его результат можно отобразить на экране ❺.

Чтобы убедиться, что отображение по умолчанию и ошибки также успешно обрабатываются новым транспортом, нужно добавить второй тест ❻, также выполняющий единственную проверку. На этот раз запрашивается отсутствующая страница (other.html), отображаемая в элементе default. Так как роль содержимого в этом элементе отображения играет пустая строка, транспорт должен сгенерировать ошибку 404. Внутри функции success проверяется ошибочная ситуация, которая не должна возникать ❼. В случае успеха данного теста должна быть вызвана функция error, в которой выполняется проверка ожидаемого кода состояния и текста сообщения об ошибке ❽. Как и в предыдущем тесте, необходимо вызвать функцию start, чтобы фреймворк QUnit смог продолжить работу после завершения вызова Ajax ❾.

12.4. Добавление преобразователя Ajax

Преобразователи дают возможность преобразовывать текстовые документы в другие форматы представления данных, более пригодные для непосредственного использования сразу после завершения вызова Ajax. В библиотеке jQuery поддерживается возможность преобразования в форматы XML и JSON с помощью функций parseXML и parseJSON соответственно. Вы также можете добавить собственные преобразователи для предварительной обработки полученных данных и преобразования их в собственные форматы.

12.4.1. Формат CSV

CSV (comma-separated values – значения, разделенные запятыми) – это распространенный текстовый формат, часто используемый для

передачи табличных данных. Каждая строка в файле CSV представляет одну запись, где значения полей отделяются запятыми (откуда и произошло название формата). Обычно первая строка в файле CSV содержит имена полей, также разделенных запятыми, и не интерпретируется как запись, например:

```
First Name,Last Name
Marcus,Cicero
Frank,Zappa
Groucho,Marx
Jane,Austen
```

Формат становится немного более сложным, когда возникает необходимость включить запятую в значение поля. Поскольку запятая обычно интерпретируется как разделитель полей, необходимо как-то показать, что внутри содержимого поля она должна интерпретироваться как обычный символ. Для этого значения полей с запятыми заключаются в кавычки ("), но тогда возникает другая проблема – проблема использования кавычек внутри значения поля. Для решения данной проблемы кавычки повторяются, и тогда пара кавычек, следующих подряд, интерпретируется как один символ кавычки.

```
First Name,Last Name,Quote
Marcus,Cicero,"""A room without books is like a body without a soul."""
Frank,Zappa,"""So many books, so little time."""
Groucho,Marx,"""Outside of a dog, a book is man's best friend. ..."""
Jane,Austen,"""The person, be it gentleman or lady, who has not ..."""
```

Из-за этих дополнительных требований к включению зарезервированных символов CSV порой бывает сложно реализовать непосредственную обработку данных на JavaScript. Чтобы упростить себе жизнь, можно было бы реализовать преобразование текстового формата CSV в соответствующий объект JavaScript со списком имен полей (`fieldsNames`) и со списком строк с данными (`rows`). Каждая строка в этом случае могла бы, в свою очередь, содержать список значений полей, соответствующих именам полей. Создание собственного преобразователя позволит вам интегрировать процедуру преобразования в стандартную инфраструктуру поддержки Ajax.

12.4.2. Преобразование текста в формат CSV

При запросе файла CSV сервером по умолчанию возвращается простая текстовая версия содержимого файла. Чтобы преобразовать этот

текст в соответствующий объект JavaScript, необходимо определить преобразователь, как показано в листинге 12.6.

Листинг 12.6. Преобразование текста с данными в формате CSV в объект

```
/* Преобразует файл CSV в объект JavaScript.
   @param  csvText  (string) текст в формате CSV
   @return (object)  объект с атрибутами, хранящими извлеченные данные
                     fieldNames (string[]) и rows (string[][]) */
// ❶ Определение функции преобразования
function textToCsv(csvText) {
    var fieldNames = [];
    var fieldCount = 9999;
    var rows = [];
    // ❷ Разбить на строки
    var lines = csvText.match(/[^\r\n]+/g);
    // ❸ Обработать каждую строку
    for (var i = 0; i < lines.length; i++) {
        if (lines[i]) {
            // ❹ Разбить на поля
            var columns = lines[i].match(/,|"([^"]|"")*"|[^,]*/g);
            var fields = [];
            var field = '';
            // ❺ Обработать каждое поле
            for (var j = 0; j < columns.length - 1; j++) {
                // Найти разделители полей
                if (columns[j] == ',') {
                    // Сохранить поле
                    if (fields.length < fieldCount) {
                        fields.push(field);
                    }
                    field = '';
                }
                else {
                    // ❻ Сохранить значение поля
                    field = columns[j].
                        replace(/^"(.*)"$/, '$1').
                        replace(/""/g, '"') || '';
                }
            }
            if (fields.length < fieldCount) { // Сохранить последнее поле
                fields.push(field);
            }
            // ❼ Установить имена полей из первой строки
            if (fieldNames.length == 0) {
                // Первая строка – заголовок
                fieldNames = fields;
                fieldCount = fields.length;
```

```
            }
            else {
                // ❽ Добавить отсутствующие поля
                for (var j = fields.length; j < fieldCount;j++){
                    fields.push('');
                }
                rows.push(fields);
            }
        }
    }
    // ❾ Вернуть объект с извлеченными данными
    return {fieldNames: fieldNames, rows: rows};
}
```

Создание нового преобразователя заключается в объявлении функции, реализующей требуемое преобразование ❶. Функции передается единственный аргумент – полный текст в формате CSV (`csvText`). За объявлением нескольких рабочих переменных следует инструкция, разбивающая файл на отдельные строки ❷ и запускающая цикл обработки каждой из них ❸. Разбиение текста выполняется передачей регулярного выражения функции `match`. Это регулярное выражение отыскивает последовательности символов, которые не являются символами перевода строки или возврата каретки (`[^\r\n]+`), при этом поиск продолжается до конца строки (флаг `g`). В результате вызова этой функции создается массив фрагментов (строк), совпавших с регулярным выражением.

Если строка не пустая, ее требуется разбить на отдельные поля ❹, учитывая при этом, что значения полей могут заключаться в кавычки. Для выполнения этой операции также используется регулярное выражение, которому соответствует одна из нескольких альтернатив (в выражении они разделяются вертикальной чертой `|`):

- запятая (`,`);
- последовательность символов, заключенная в двойные кавычки (`"([^"]|"")*"`), которая может включать экранированные двойные кавычки (`""`);
- последовательность символов, не содержащая запятую (`[^,]*`).

Поиск этих последовательностей выполняется на протяжении всей строки (флаг `g`). Получившийся в результате массив будет содержать значения полей (возможно, заключенные в кавычки), а также запятые, разделяющие поля.

Далее выполняется обход всех элементов массива ❺. Если текущий элемент является разделителем-запятой, в список полей (`fields`)

добавляется поле (field), найденное в предыдущей итерации, но только если текущее количество полей меньше количества полей, найденных в первой строке в файле CSV (где определяются имена полей). Если текущий элемент не является запятой, его содержимое запоминается в переменной field ❻ после удаления из него окружающих кавычек и преобразования каждой экранированной кавычки в единственный символ. Это необходимо сделать, чтобы избавиться от дополнительных символов, которые были добавлены из-за присутствия запятых в значениях полей. После обработки всех совпадений, найденных в строке, выполняется сохранение значения последнего поля.

Если это – первая строка в файле CSV (то есть если прежде имена полей еще не сохранялись), значения полей переносятся в список имен полей (fieldNames) ❼. В противном случае ненайденные поля заполняются пустыми строками до достижения их количества, равного количеству имен полей ❽, и текущая строка добавляется в список уже обработанных строк (rows).

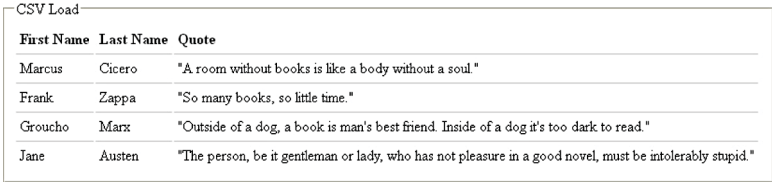
После обработки всех строк сценарию возвращается список извлеченных имен полей и строк в виде объекта JavaScript ❾.

> Более полную реализацию преобразования данных в формате CSV можно найти в расширении jQuery CSV по адресу: https://code.google.com/p/jquery-csv/.

Зарегистрировать новый преобразователь можно с помощью функции $.ajaxSetup, которой нужно передать объект converters, содержащий атрибут, имя которого совпадает с именами исходного и конечного типов данных, а значением является ссылка на функцию преобразования:

```
$.ajaxSetup({converters: {
    'text csv': textToCsv
}});
```

Чтобы задействовать преобразователь, в вызове функции ajax следует указать в параметре dataType тип данных csv. Указанной вами функции обратного вызова success будет передан объект с результатами преобразования, которые вы сможете обрабатывать непосредственно, не выполняя промежуточного парсинга текстового формата CSV. На рис. 12.3 показан результат загрузки данных в формате CSV в таблицу, полученный с помощью кода из листинга 12.7.


CSV Load

First Name	Last Name	Quote
Marcus	Cicero	"A room without books is like a body without a soul."
Frank	Zappa	"So many books, so little time."
Groucho	Marx	"Outside of a dog, a book is man's best friend. Inside of a dog it's too dark to read."
Jane	Austen	"The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid."

Рис. 12.3 ❖ Данные в формате CSV загружены в таблицу

Листинг 12.7. Загрузка данных в формате CSV

```
// ❶ Запросить и преобразовать данные в формате CSV
$.ajax({url: 'quotes.csv', dataType: 'csv', success: function(csv) {
    // ❷ Создать заголовок таблицы
    var table = '<table><thead><tr>';
    for (var i = 0; i < csv.fieldNames.length; i++) {
        table += '<th>' + csv.fieldNames[i] + '</th>';
    }
    table += '</tr></thead><tbody>';
    // ❸ Обработать каждую строку
    for (var i = 0; i < csv.rows.length; i++) {
        table += '<tr>';
        // ❹ Создать строки в таблице
        for (var j = 0; j < csv.fieldNames.length; j++) {
            table += '<td>' + csv.rows[i][j] + '</td>';
        }
        table += '</tr>';
    }
    table += '</tbody></table>';
    // ❺ Добавить таблицу в страницу
    $('#tableResult').append(table);
}});
```

Чтобы загрузить файл CSV, необходимо передать функции `ajax` адрес URL файла и параметр настройки `dataType` со значением `csv` ❶. Обратите внимание, что если подключить к странице предварительный фильтр из раздела 12.2.1, вам не придется указывать тип данных, так как он будет определяться автоматически по расширению файла в строке URL.

Когда библиотека jQuery загрузит файл, ей потребуется преобразовать его из формата по умолчанию (`text`) в запрошенный, поэтому она отыщет зарегистрированный преобразователь и применит его. В результате преобразования будет получен объект JavaScript с данными, извлеченными из файла CSV, который будет передан функции обратного вызова `success` для дальнейшей обработки.

Внутри функции обратного вызова создается новая таблица в виде строкового значения (table), начиная с ячеек в заголовке, содержащих названия полей ❷. Затем выполняется обработка строк ❸, в ходе которой в таблицу добавляются ячейки со значениями полей ❹. В заключение новая таблица добавляется в страницу ❺.

12.4.3. Преобразование данных CSV в таблицу

Так как потребность представления данных CSV в табличной форме возникает достаточно часто, процесс преобразования можно развить еще дальше и реализовать преобразование объекта, полученного в листинге 12.7, в таблицу, чтобы ее можно было встраивать в страницу непосредственно. Такое преобразование выполняет функция, представленная в листинге 12.8.

Листинг 12.8. Преобразование объекта CSV в таблицу

```
/* Преобразование JavaScript-объекта в формате CSV в HTML-таблицу.
   @param  csv  (object) объект CSV
   @return (jQuery) данные в таблице */
// ❶ Определение функции преобразования
function csvToTable(csv) {
    // ❷ Сгенерировать таблицу для данных CSV
    var table = '<table><thead><tr>';
    for (var i = 0; i < csv.fieldNames.length; i++) {
        table += '<th>' + csv.fieldNames[i] + '</th>';
    }
    table += '</tr></thead><tbody>';
    for (var i = 0; i < csv.rows.length; i++) {
        table += '<tr>';
        for (var j = 0; j < csv.fieldNames.length; j++) {
            table += '<td>' + csv.rows[i][j] + '</td>';
        }
        table += '</tr>';
    }
    table += '</tbody></table>';
    // ❸ Вернуть элемент созданной таблицы
    return $(table);
}
```

Как и прежде, преобразование будет выполнять отдельная функция, но в данном случае – преобразующая объект с данными CSV в таблицу HTML ❶. Тело этой функции ❷ идентично телу функции обратного вызова success, использовавшейся в предыдущем примере применения преобразователя (листинг 12.7). Таблица генерируется в виде строкового значения в процессе обхода, сначала ячеек с названиями полей, а затем строк с фактическими данными. Полученный

в результате текст используется как основа для создания элемента DOM, который возвращается вызывающему коду ❸.

Зарегистрировать новый преобразователь можно с помощью функции `$.ajaxSetup`, которой нужно передать объект `converters`, содержащий всю необходимую информацию. Обратите внимание, что роль исходного типа играет объект CSV, создаваемый предыдущим преобразователем:

```
$.ajaxSetup({converters: {
    'csv table': csvToTable
}});
```

Теперь код, загружающий файл CSV и преобразующий его в таблицу, будет выглядеть значительно короче. Как и прежде, нужно вызвать функцию `ajax` и передать ей адрес URL файла CSV. Но, в отличие от предыдущего примера, значение параметра типа данных должно определяться в виде последовательности из двух типов: первый будет использоваться для преобразования текста в объект, а второй – для преобразования объекта в таблицу. Обратите внимание, что если подключить к странице предварительный фильтр из раздела 12.2.1, вам не придется указывать первый тип данных в последовательности, так как он будет определяться автоматически по расширению файла в строке URL. Получающийся в результате элемент `table` передается как аргумент функции обратного вызова `success` и может быть добавлен в страницу непосредственно:

```
$.ajax({url: 'quotes.csv', dataType: 'csv table',
    success: function(table) {
        $('#tableResult').append(table);
    }
});
```

Возможность создания собственных преобразователей позволяет последовательно трансформировать данные из одного формата в другой, начиная с простого текста, возвращаемого механизмом Ajax, и, возможно, через несколько промежуточных преобразований, и заканчивая форматом, наиболее подходящим для решения текущей задачи.

12.5. Расширения Ajax

Расширения Ajax, описанные в этой главе, доступны для загрузки на веб-сайте книги, где также можно найти страницу, демонстрирующую работу различных расширений с инфраструктурой Ajax.

Это нужно знать

Библиотека jQuery упрощает доступ к удаленным ресурсам, предоставляя функцию `ajax` и другие родственные ей функции.

Создавайте расширения Ajax, когда требуется удовлетворить дополнительные требования к удаленным взаимодействиям и форматам данных.

Имеется возможность дополнить своими операциями или вообще отменить запрос, зарегистрировав предварительный фильтр с помощью функции `$.ajaxPrefilter`.

Функция `$.ajaxTransport` используется для регистрации новых механизмов извлечения удаленного содержимого.

Дополнительные средства преобразования данных можно регистрировать, передавая их в аргументе `converters` функции `$.ajaxSetup`.

Поддерживается возможность объединения типов данных в цепочки для преобразования данных в конечный формат путем последовательного выполнения нескольких промежуточных этапов.

Попробуйте сами

Создайте новый преобразователь, напоминающий пример реализации преобразования CSV в таблицу, который преобразовывал бы объект CSV в список. Каждый элемент этого списка должен содержать несколько строк с метками, соответствующими названиям полей, за которыми следуют значения соответствующих полей в текущей записи. Зарегистрируйте новый преобразователь и примените его к примеру с исходными данными, использовавшемуся выше.

12.6. В заключение

Библиотека jQuery упрощает использование технологии Ajax, скрывая тонкости работы с объектом `XMLHttpRequest` за простым в использовании интерфейсом. Функция `ajax` обеспечивает полный контроль над процессом выполнения запроса, но она не всегда удобна в использовании, однако имеется множество родственных ей вспомогательных функций, значительно упрощающих взаимодействия. Когда запрос запускается на выполнение, инфраструктура поддержки Ajax сначала проверяет наличие предварительных фильтров, которые можно было бы применить к данному запросу, чтобы, возможно, изменить его или даже отменить. Далее выбирается механизм транспорта, способный обрабатывать данные запрошенного формата и загружать их. Наконец, для трансформации полученного содержимого в более подходящий формат может быть вызван преобразователь.

В процесс обработки запросов и данных можно внедриться на любом из перечисленных этапов. Добавьте предварительный фильтр, чтобы настроить доступ к удаленному содержимому или вообще запретить доступ, как было показано в этой главе. Реализуйте в виде

собственной функции альтернативный механизм для получения особого содержимого, например для загрузки изображений. Замените или дополните существующий механизм, как это было сделано в примере реализации имитации загрузки HTML для нужд тестирования. Получайте данные в наиболее удобных форматах, интегрируя свои преобразователи в инфраструктуру поддержки Ajax. Все вместе эти точки интеграции дают вам возможность реализовать любую обработку данных в процессе выполнения запросов Ajax.

В следующей главе рассматриваются вопросы, связанные с обработкой событий, и исследуются возможности определения собственных событий.

Глава 13

Расширение поддержки событий

Эта глава охватывает следующие темы:

- инфраструктура поддержки событий в библиотеке jQuery;
- добавление специализированных событий;
- расширение существующих событий.

Библиотека jQuery упрощает подключение обработчиков событий к любым стандартным событиям, возникающим в процессе работы веб-страницы, таким как щелчок мышью и нажатие клавиши. Для этой цели, помимо специализированных функций, работающих с коллекциями элементов, таких как `click` и `keyup`, можно использовать универсальные функции `bind` и `on`, выполняющие подключение обработчиков к именованным событиям.

Библиотека jQuery расширяет стандартный механизм обработки событий, добавляя в него возможность подключения нескольких обработчиков к одному и тому же событию, генерируемому элементом, и поддержку пространств имен, чтобы упростить различение этих обработчиков. Она также обеспечивает возможность делегирования событий, то есть позволяет подключить обработчик события к контейнерному элементу, но вызывает его, когда событие возникает в одном из вложенных элементов, что позволяет уменьшить количество зарегистрированных обработчиков. Кроме того, механизм делегирования событий дает возможность определять обработчики для элементов, отсутствующих в дереве DOM на момент подключения.

Зачастую внутри страницы происходит масса событий, не нашедших отражения в стандартной системе событий JavaScript, например активация (enabling) и деактивация (disabling) элементов управления. Поддержка подобных событий на манер стандартных могла бы принести определенные удобства. Для таких ситуаций jQuery предлагает собственную *инфраструктуру поддержки специализирован-*

ных событий. Специализированные события могут предусматривать собственные процедуры инициализации и финализации, выполняемые при подключении и отключении обработчиков от элементов. Они могут изменять сгенерированные события или даже генерировать совершенно новые события. С помощью этой инфраструктуры можно определять собственные события, связанные (необязательно) со стандартными событиями JavaScript, и интегрировать их в стандартный механизм обработки событий, сохраняя возможность использовать все функциональные возможности jQuery.

Для обеспечения единообразия поддержки событий в разных браузерах jQuery использует собственную инфраструктуру событий, благодаря чему можно использовать единый подход к обработке как стандартных, так и собственных событий. В этой главе мы создадим событие щелчка правой кнопкой мыши сначала в виде самостоятельного события, а затем интегрируем его в стандартную процедуру `click`. Дополнительно вы узнаете, как запретить это событие для отдельных элементов страницы и как обслуживать несколько щелчков в единственном событии.

13.1. Инфраструктура поддержки специализированных событий

Инфраструктура специализированных событий в библиотеке jQuery дополняет встроенный механизм событий JavaScript. Она позволяет определять собственные события со специфическим поведением и даже видоизменять стандартные события, добавляя в них новые функциональные возможности.

Специализированное событие, как ожидается, является совершенно новым событием, реализует собственные процедуры инициализации и финализации и имеет внутренний обработчик события. Однако точно так же можно расширить стандартное событие и делегировать слежение за ним библиотеке jQuery.

Инфраструктура используется также самой библиотекой для создания единой системы событий, таких как `mouseenter` и `mouseleave`, одинаково работающих в разных браузерах. Эти два события генерируются, только когда указатель мыши пересекает границы выбранных элементов, тогда как стандартные события `mouseover` и `mouseout` дополнительно генерируются при пересечении границ любых вложенных элементов.

В этом разделе вы увидите, как с помощью инфраструктуры специализированных событий определять обработчики для новых событий, как она вызывает их при попытке пользователя связать событие нового типа с элементом и повторно вызывает их при возбуждении самого события.

13.1.1. Подключение обработчиков событий

Чтобы задействовать инфраструктуру специализированных событий, необходимо зарегистрировать объект обработчика события, содержащий функции и/или атрибуты, переопределяющие порядок обработки события, принятый по умолчанию. jQuery будет использовать этот объект для работы с соответствующими событиями.

На рис. 13.1 изображена диаграмма операций, выполняемых при попытке пользователя подключить или отключить собственный обработчик специализированного события.

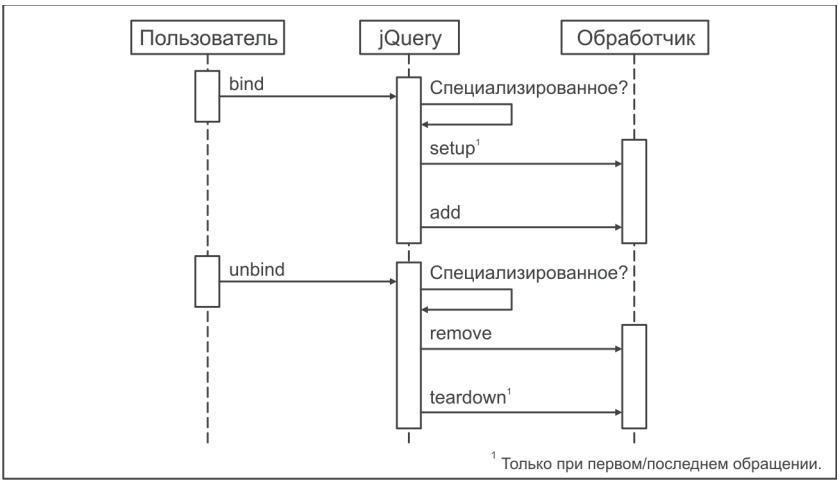


Рис. 13.1 ❖ Диаграмма процедуры подключения и отключения специализированных событий

Когда пользователь подключает специализированное событие к элементу (с использованием функции `bind` или `on`), jQuery проверяет наличие объекта-обработчика для события этого типа. Если специальный объект-обработчик отсутствует, jQuery будет использовать стандартный механизм JavaScript для обработки этого события. Если

специальный объект-обработчик имеется, вызывается его функция `setup`, но только если событие этого типа подключается к текущему элементу впервые. Функция `add` объекта-обработчика вызывается всегда – она обеспечивает настройку каждой пользовательской функции-обработчика данного события для данного элемента.

В зависимости от уровня настройки, допустимого для ваших событий, одна из этих двух функций установит обработчик, откликающийся на события, который впоследствии будет вызывать пользовательские функции-обработчики в соответствующие моменты времени.

Аналогично, когда пользователь предпримет попытку отключить специализированное событие от элемента (с использованием функции `unbind` или `off`), jQuery снова проверит наличие объекта-обработчика и перейдет на использование стандартного механизма при его отсутствии. Если jQuery найдет объект-обработчик, она будет вызывать его функцию `remove` при каждой попытке пользователя отключить собственную функцию-обработчик от элемента. После отключения последней функции-обработчика будет вызвана функция `teardown` объекта. В этой функции можно реализовать операции, отменяющие все изменения, произведенные в функции `setup`.

13.1.2. Возбуждение событий

После связывания события, лежащего в основе вашего специализированного события, с помощью функции `setup` или `add`, как описывалось выше, вам остается только сидеть и ждать, когда это событие произойдет.

На рис. 13.2 показано, что происходит в системе, когда возникает базовое событие.

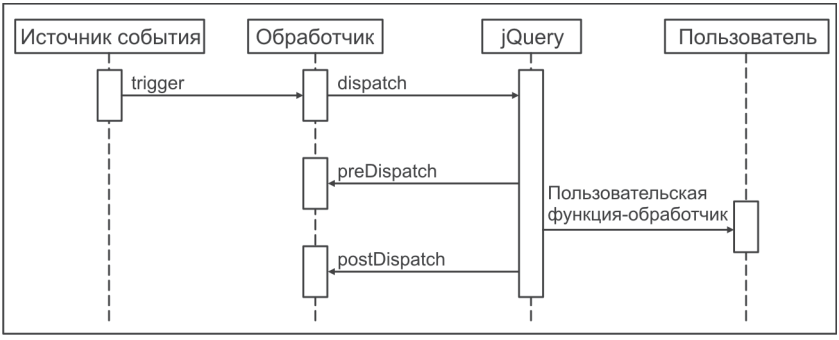


Рис. 13.2 ❖ Диаграмма процедуры обработки событий

Независимо от используемого механизма возбуждения события вызывается внутренний обработчик, находящийся в объекте-обработчике. После выполнения дополнительной логики, определяющей, например, в ответ на какое количество щелчков мыши должно генерироваться событие, объект события обновляется и передается механизму диспетчеризации событий в библиотеке jQuery.

jQuery снова проверит, имеется ли для события данного типа специализированный объект-обработчик, и вызовет функцию `preDispatch` в этом объекте, чтобы выполнить дополнительные предварительные операции, включая прерывание процесса обработки события. Затем будет вызвана пользовательская функция-обработчик, которая получит объект события, созданный внутренним обработчиком. И наконец, jQuery вызовет функцию `postDispatch` объекта-обработчика для выполнения заключительных операций.

Чтобы увидеть, как все это работает в действительности, создадим специализированное событие щелчка правой кнопкой мыши.

13.2. Добавление специализированного события

Допустим, что нам требуется выполнить некоторые действия, когда пользователь щелкает правой кнопкой мыши. Теоретически можно было бы подключить обработчик события `click` и проверять, какой кнопкой был выполнен щелчок (анализируя атрибут `event.which`). Но браузеры перехватывают событие щелчка правой кнопкой мыши, чтобы вывести собственное контекстное меню, и не дают возможности обработать это событие в сценарии на JavaScript. Эту проблему можно решить, определив обработчик специализированного события, как описывается в данном разделе. После этого у вас появится возможность регистрировать собственные функции-обработчики события щелчка правой кнопкой мыши, как это делается для события щелчка левой кнопкой.

Чтобы добавить собственное событие, необходимо подключить соответствующий объект-обработчик к точке интеграции `$.event.special`. Обычно объект-обработчик определяет функцию `setup`, позволяющую инициализировать обработку события в элементе, и функцию `teardown`, отменяющую все изменения, выполненные функцией `setup`.

Когда пользователь попытается подключить свою функцию обратного вызова для обработки этого нового события, инфраструкту-

ра событий в библиотеке jQuery будет использовать данный объект для вызова соответствующих функций в нужные моменты времени. Благодаря этому вы сможете использовать все преимущества инфраструктуры, такие как пространства имен, делегирование событий и средства управления их распространением.

13.2.1. Добавление события щелчка правой кнопкой мыши

Браузеры перехватывают событие щелчка правой (и средней) кнопкой мыши для собственных нужд, как же тогда перехватить это событие, чтобы получить возможность обрабатывать его самостоятельно? Дело в том, что в данной ситуации генерируется событие с именем `contextmenu`. Определив собственное специальное событие, мы сможем присвоить ему более понятное имя `rightclick` и добавить в него дополнительную функциональность.

Подключить новое событие непосредственно к выбранным элементам можно с помощью стандартной функции `bind` или `on`, и получить в результате возможность обрабатывать его, как показано на рис. 13.3.

```
$('#myDiv').on('rightclick', function(event) {
    alert('Notified of event ' + event.type);
});
```

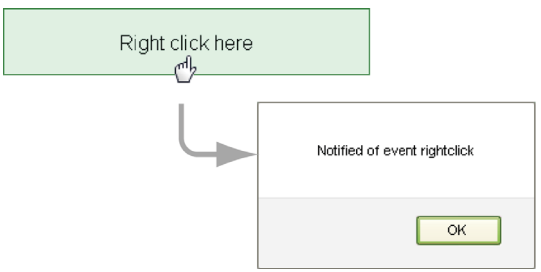


Рис. 13.3 ❖ Элемент div, определяющий нестандартный внешний вид указателя мыши и реализующий вывод диалога с сообщением в ответ на щелчок правой кнопкой мыши

В листинге 13.1 приводится определение объекта-обработчика специализированного события щелчка правой кнопкой мыши. Как обычно, определение заключено в вызов анонимной функции, которая создает новую область видимости для переменных и гаранти-

рует возможность использования идентификатора $ для ссылки на объект jQuery.

Листинг 13.1. Добавление события щелчка правой кнопкой мыши

```
(function($) { // Скрывает из видимости внешнего кода,
               // устраняет конфликты с идентификатором $

/* Оределение события щелчка правой кнопкой мыши. */
// ❶ Определение объекта-обработчика специализированного события
$.event.special.rightclick = {

    /* ❷ Тип нового события. */
    eventType: 'rightclick',

    /* Инициализирует обработчик события щелчка правой кнопкой мыши.
       @param data (object, необязательный) любые данные,
                                            переданные в вызов bind
       @param namespaces (string[]) любые пространства имен,
                                    переданные в вызов bind */
    // ❸ Выполняет инициализацию при подключении первого обработчика
    setup: function(data, namespaces) {
        $(this).addClass('right-clickable').
            bind('contextmenu', $.event.special.rightclick.handler);
    },

    /* Удаляет обработчик события щелчка правой кнопкой мыши.
       @param namespaces (string[]) любые пространства имен,
                                    переданные в вызов unbind */
    // ❹ Выполняет заключительные операции
    teardown: function(namespaces) {
        $(this).removeClass('right-clickable').
            unbind('contextmenu', $.event.special.rightclick.handler);
        },

    /* Реализует фактическую обработку события.
       @param event (Event) объект с информацией о событии
       @return (boolean) false - чтобы подавить поведение по умолчанию */
    // ❺ Внутренний обработчик события
    handler: function(event) {
        event.type = $.event.special.rightclick.eventType;
        // ❻ Вызов пользовательской функции-обработчика
        return $.event.dispatch.apply(this, arguments);
    }
};
})(jQuery);
```

Чтобы определить новое событие, необходимо к точке интеграции $.event.special подключить объект с именем этого события ❶. В соответствии с соглашениями имена событий конструируются только

из символов нижнего регистра. Хотя это и необязательно, все же нелишним будет включить в объект атрибут, хранящий имя события ❷, чтобы его можно было единообразно использовать внутри расширения.

Функция `setup` ❸ вызывается один раз, при первом подключении события данного типа к элементу, позволяя однократно выполнить инициализацию всего, что необходимо для поддержки этого события. Она принимает два аргумента: переданные в вызов функции `bind` или `on` любые данные (`data`) и пространства имен (`namespaces`). Если вместе с событием никаких данных передавать не требуется, этот аргумент будет иметь значение `undefined`. Если событие не связано с каким-то пространством имен, в аргументе `namespaces` будет передан массив с единственной пустой строкой. Внутри функции переменная `this` будет ссылаться на целевой элемент.

При подключении события `rightclick` можно добавить в целевой элемент класс CSS для настройки его внешнего вида, чтобы показать, что он доступен для щелчка правой кнопкой мыши. Например, для элементов с этим классом можно изменять внешний вид указателя мыши при его наведении. Необходимо также зарегистрировать внутренний обработчик события, вызываемый в ответ на стандартное событие `contextmenu`, которое мы собираемся преобразовывать в событие `rightclick`.

Функция `teardown` ❹ позволяет отменить все изменения, произведенные в функции `setup` на этапе инициализации. Она также вызывается однократно, при отключении от элемента последнего пользовательского обработчика события данного типа. Данная функция тоже принимает список пространств имен событий. Для события `rightclick` необходимо удалить класс CSS и отключить обработку события `contextmenu`.

Внутренний обработчик, зарегистрированный для обработки события `contextmenu` ❺, выполняет преобразование типа события (передается в виде аргумента) перед передачей его подключенным пользовательским обработчикам посредством функции `$.event.dispatch` ❻. Несмотря на то что функции `setup` и `teardown` вызываются только один раз для каждого элемента, сохраняется возможность зарегистрировать несколько пользовательских обработчиков для одного и того же типа события, и все они будут вызваны в ходе обращения к этой функции.

Дополнительно предоставляется возможность запрещать передачу данного события для отдельных элементов.

13.2.2. Запрет передачи события щелчка правой кнопкой мыши

Иногда бывает желательно запретить передачу события щелчка правой кнопкой мыши для некоторых или всех элементов, например чтобы исключить возможность обработки двойных щелчков. Так как теперь имеется полный контроль над обработкой событий этого типа, вы легко сможете реализовать данное ограничение, как показано в листинге 13.2.

Листинг 13.2. Запрет передачи события щелчка правой кнопкой мыши

```
// ❶ Список элементов, для которых запрещено генерировать событие
$.event.rightclickDisabled = $([]);

/* Определение события щелчка правой кнопкой мыши. */
$.event.special.rightclick = {

    ...

    /* Реализует фактическую обработку события.
       @param event (Event) объект с информацией о событии
       @return (boolean) false - чтобы подавить поведение по умолчанию */
    handler: function(event) {
        // ❷ Если данный элемент присутствует в списке...
        if ($.event.rightclickDisabled.length &&
                $(this).is($.event.rightclickDisabled)) {
            // ❸ ...прервать дальнейшую его обработку
            event.stopPropagation();
            event.preventDefault();
            return;
        }
        event.type = $.event.special.rightclick.eventType;
        return $.event.dispatch.apply(this, arguments);
    }
};
```

Здесь определяется переменная, хранящая список элементов, для которых запрещено генерировать событие ❶. Эта переменная должна быть доступна за пределами расширения, поэтому ее желательно подключить где-то в области точки интеграции `$`, лучше в точке `$.event`, чтобы лишний раз показать, что она имеет отношение к событиям. Переменная инициализируется пустой коллекцией jQuery. Ее можно замещать или добавлять в нее новые элементы. Добавить элемент в коллекцию можно с помощью функции `add`:

```
$.event.rightclickDisabled = $.event.rightclickDisabled.add('#elemID')
```

Удалить из коллекции – с помощью функции filter:

```
$.event.rightclickDisabled =
    $.event.rightclickDisabled.filter(':not(#elemID)')
```

Реализация новой функциональности добавлена в начало функции handler. Здесь проверяется присутствие текущего элемента в списке (на него ссылается переменная this) с помощью функции is ❷. Если элемент найден, вызываются функции события stopPropagation и preventDefault, чтобы остановить обработку события в этой точке ❸.

Событие, созданное в данном разделе, генерируется в ответ на один щелчок правой кнопкой мыши, а что, если потребуется генерировать событие в ответ на два или более щелчков правой кнопкой? Пример определения такого необычного события как раз рассматривается в следующем разделе.

13.2.3. Событие многократных щелчков правой кнопкой

Реализовать событие многократных щелчков правой кнопкой мыши значительно сложнее, чем событие одного щелчка, из-за необходимости следить за количеством щелчков. Кроме того, следует предусмотреть возможность настройки количества щелчков, в ответ на которое будет возбуждаться событие, и максимальной продолжительности временного интервала между щелчками, чтобы они рассматривались как принадлежащие одному событию.

С помощью этого события можно реализовать свою реакцию на множественные щелчки мышью. Более того, в ответ на разное количество щелчков можно даже генерировать разные события, как показано на рис. 13.4, где видно, что в ответ на двойные и тройные щелчки выводятся разные сообщения (реализация пользовательского обработчика приводится в листинге 13.3).

Листинг 13.3. Обработка событий двойных и тройных щелчков мышью

```
$('#multiRightClick').
    bind('rightmulticlick', function(event) {
        $('#multiOutcome').val($('#multiOutcome').val() +
            'Right-double-clicked\n');
        return false;
    }).
    bind('rightmulticlick', {clickCount: 3}, function(event) {
        $('#multiOutcome').val($('#multiOutcome').val() +
            'Right-triple-clicked\n');
        return false;
    });
```

Мы могли бы расширить предыдущее специализированное событие и добавить в него поддержку множественных щелчков, но давайте поступим иначе и создадим новый тип события, чтобы получить возможность сравнить их. Определение нового события приводится в листинге 13.4.

Листинг 13.4. Событие множественного щелчка

```
/* Оределение события щелчка правой кнопкой мыши. */
// ❶ Определение объекта-обработчика специализированного события
$.event.special.rightmulticlick = {

    /* ❷ Тип нового события. */
    eventType: 'rightmulticlick',

    /* Инициализирует обработчик события множественного
       щелчка правой кнопкой мыши.
       @param data (object, необязательный) любые данные,
                                            переданные в вызов bind
       @param namespaces (string[]) любые пространства имен,
                                    переданные в вызов bind */
    // ❸ Выполняет инициализацию при подключении первого обработчика
    setup: function(data, namespaces) {
        $(this).addClass('right-clickable');
    },

    /* Инициализирует параметры настройки.
       @param handleObj (object) параметры настройки */
    // ❹ Инициализация параметров настройки для каждого обработчика
    add: function(handleObj) {
        // ❺ Данные, характерные для события
        var data = $.extend({clickCount: 2, clickNumber: 0,
            lastClick: 0, clickSpeed: 500,
            handler: handleObj.handler}, handleObj.data || {});
            var id = $.event.special.rightmulticlick.eventType +
                handleObj.guid;
            // ❻ Сохранить данные и подключить
            //    внутренний обработчик событий
            $(this).data(id, data).
                bind('contextmenu.' + id, {id: id},
                    $.event.special.rightmulticlick.handler);
    },

    /* Сбрасывает настройки.
       @param handleObj (object) параметры настройки */
    // ❼ Заключительные операции после удаления каждого обработчика
    remove: function(handleObj) {
        var id = $.event.special.rightmulticlick.eventType +
            handleObj.guid;
```

```
        // ❽ Удаляет данные и отключает внутренний обработчик событий
        $(this).removeData(id).unbind('contextmenu.' + id);
    },

    /* Выполняет заключительные операции.
       @param namespaces (string[])любые пространства имен,
                                   переданные в вызов unbind*/
    // ❾ Выполняет заключительные операции
    teardown: function(namespaces) {
        $(this).removeClass('right-clickable');
    },
    ...
};
```

Как и в предыдущем примере, объект-обработчик нового специализированного события подключается к точке интеграции `$.event.special` ❶ и содержит атрибут, хранящий тип события ❷. В функции `setup` ❸ требуется только добавить класс CSS для нужд оформления элементов. Так как каждый обработчик, подключаемый пользователем, может определять собственные параметры настройки, у нас уже не получится использовать единственный обработчик для всех событий.

Поэтому требуется определить функцию `add` ❹, которая будет вызываться при каждой попытке подключить к элементу обработчик событий данного типа, и обрабатывать каждый обработчик отдельно. Функция принимает единственный аргумент (`handleObj`), заключающий в себе всю информацию о настройках текущего обработчика событий. Этот объект имеет атрибуты для хранения произвольных данных (`handleObj.data`), названия пространства имен (`handleObj.namespace`), пользовательской функции-обработчика (`handleObj.handler`), уникального идентификатора (`handleObj.guid`) и др. Как уже говорилось выше, переменная `this` ссылается на текущий целевой элемент.

Добавление нового обработчика начинается с объединения параметров настройки для этого обработчика ❺, переданных в вызов функции `bind` или `on`, с параметрами по умолчанию. Так как пользователь может не передавать свои данные при регистрации, по умолчанию вместо них следует сохранить пустой объект (`|| {}`).

Чтобы позднее можно было извлечь параметры настройки, на основе типа события и уникального идентификатора в параметрах настройки создается идентификатор обработчика события. Он используется для связывания параметров с текущим элементом (`this`) ❻ и обработчика событий, имя которого содержит название пространства

имен, со стандартным событием `contextmenu`, при этом идентификатор передается обработчику как дополнительные данные. Сам обработчик будет описан далее.

Когда обработчик отключается от элемента, вызывается функция `remove` ❼. Она также получает объект с настройками события (`handleObj`). Эта функция повторно вычисляет идентификатор на основе типа события и уникального идентификатора в параметрах настройки и использует его для удаления сохраненных параметров и отключения обработчика события с именем, включающим идентификатор в виде пространства имен ❽.

Чтобы отменить все изменения, выполненные функцией `setup`, функции `teardown` ❾ достаточно всего лишь удалить добавленный класс CSS.

Как и в случае с событием единственного щелчка правой кнопкой, внутренний обработчик вызывается в ответ на событие `contextmenu` и преобразует его в ваше событие. Но на этот раз дополнительно нужно отследить количество выполненных щелчков, как показано в листинге 13.5.

Листинг 13.5. Обработчик события множественного щелчка правой кнопкой

```
// ❶ Внутренний обработчик события
$.event.special.rightmulticlick = {
    ...

    /* Реализует фактическую обработку события.
       @param event (Event) объект с информацией о событии
       @return (boolean) false - чтобы подавить поведение по умолчанию */
    handler: function(event) {
        // ❷ Игнорировать, если обработка запрещена
        if ($.event.rightclickDisabled.length &&
                $(this).is($.event.rightclickDisabled)) {
            event.preventDefault();
            return;
        }
        // ❸ Извлечь параметры настройки события
        var data = $(this).data(event.data.id);
        // ❹ Получить текущее время
        event.timeStamp = event.timeStamp || new Date().getTime();
        // ❺ Если максимально допустимый интервал не превышен,
        //    увеличить счетчик
        if (event.timeStamp - data.lastClick <= data.clickSpeed) {
            data.clickNumber++;
        }
        // ❻ Иначе сбросить счетчик
```

```
        else {
            data.clickNumber = 1;
            data.lastClick = event.timeStamp;
        }
        var result = false;
        // ❼ Если требуемое число достигнуто...
        if (data.clickNumber == data.clickCount) {
            event.type = $.event.special.rightmulticlick.eventType;
            // ❽ Вызвать пользовательский обработчик события
            result = data.handler.apply(this, arguments);
        }
        // ❾ Вернуть результат
        return result;
    }
};
```

Определяемая функция-обработчик ❶ вызывается для обработки стандартного события `contextmenu`. Как и в обработчике единственного щелчка правой кнопкой, следует сначала проверить, не была ли запрещена обработка данного события в элементе, и при необходимости предотвратить дальнейшее его распространение ❷. Если обработка события разрешена, обработчик извлекает параметры настройки для текущего элемента (`this`) ❸, используя идентификатор, переданный вместе с данными при привязке пользовательской функции обратного вызова (`event.data.id`).

Чтобы ограничить время, в течение которого множественные щелчки рассматриваются как единое событие, необходимо сначала определить время события ❹, используя либо переданное значение, либо текущие дату и время (выражается в миллисекундах, прошедших с некоторого базового момента). Если разница между моментами времени текущего и предыдущего событий (хранится в атрибуте `data.lastClick`) меньше или равна указанному ограничению (`data.clickSpeed`), увеличивается счетчик щелчков ❺. Если предельно допустимый интервал времени превышен, необходимо сбросить счетчик щелчков и установить момент времени начала новой последовательности ❻.

Далее количество выполненных щелчков сравнивается с требуемым (`data.clickCount`) ❼. Если они равны, принятое событие преобразуется в событие нового типа и вызывается пользовательский обработчик этого события (`data.handler`) ❽. Так как зарегистрированные обработчики могут иметь разные настройки, здесь нельзя использовать функцию `$.event.dispatch`, как в примере с событием единственного щелчка правой кнопкой. Результат вызова пользовательского

обработчика сохраняется и возвращается в виде возвращаемого значения внутреннего обработчика ❾.

Для удобства пользователей можно также реализовать функции коллекций, подключающие ваши специализированные события, о чем мы и поговорим в следующем разделе.

13.2.4. Функции для взаимодействия с событиями

Реализовав поддержку специализированного события, вы получаете возможность подключать обработчики этого события к элементам с помощью функций `bind` и `on` (и отключать с помощью функций `unbind` и `off`). Применение этих функций дает возможность пользоваться другими особенностями инфраструктуры поддержки событий, такими как пространства имен, делегирование событий и управление их распространением. Но было бы гораздо удобнее, если бы для простых случаев имелись вспомогательные функции, выполняющие подключение или генерирующие событие программно, как это делает функция `click` в отношении обычного события щелчка мышью.

Наличие таких функций позволит пользователям выбирать наиболее удобный для них способ взаимодействия с вашим событием – использовать функции `bind`/`on`/`trigger` или функции, имена которых совпадают с именами событий, как это позволяют встроенные события.

```
$('#myDiv').rightclick(function() {
    ... // Обработать щелчок правой кнопкой
});
```

Такие функции, которые фактически являются расширениями коллекций, определяются, как показано в листинге 13.6.

Листинг 13.6. Добавление функций коллекций для взаимодействия с событиями

```
/* Добавление функций коллекций для взаимодействия с событиями. */
// ❶ Обработать типы событий
$.each([$.event.special.rightclick.eventType,
        $.event.special.rightmulticlick.eventType],
    function(i, eventType) {
        // ❷ Создать функцию коллекций
        $.fn[eventType] = function(data, fn) {
            // ❸ Обработать ситуацию, когда данные не были переданы
            if (fn == null) {
                fn = data;
                data = null;
            }
```

```
            // ❹ Подключить или вызвать обработчик
            return arguments.length > 0 ?
                this.on(eventType, null, data, fn) :
                this.trigger(eventType);
        };
    }
);
```

Чтобы упростить взаимодействия с событиями, необходимо зарегистрировать функции для всех типов событий, указанных во встроенном массиве ❶. Для каждого события определяется новое расширение коллекций (подключается к точке расширения `$.fn`) с именем, совпадающим с типом события ❷, принимающее два аргумента: данные для передачи обработчикам событий (`data`, необязательно) и функцию для вызова по событию (`fn`).

Внутри расширения требуется обработать ситуацию, когда оно вызывается без аргумента `data` (то есть когда второй аргумент отсутствует) ❸, – в этом случае первый аргумент интерпретируется как ссылка на функцию-обработчик события, а аргумент `data` очищается. Если количество аргументов, переданных функции расширения, больше нуля ❹, указанную функцию `fn` следует подключить к текущему элементу как обработчик события данного типа. В противном случае следует возбудить событие и тем самым вызвать все имеющиеся обработчики этого события, как это делают стандартные функции взаимодействий с событиями.

13.3. Расширение существующих событий

Помимо определения новых событий, существует также возможность замещать или расширять существующие события, определяя специализированные события с теми же именами. Например, можно расширить существующее событие `click` так, чтобы каждый элемент, поддерживающий это событие, изменял вид указателя мыши, как предложено Дэвидом Уолшем (David Walsh) и реализовано Брендоном Аароном (Brandon Aaron)[1]. Можно также реализовать регистрацию каждого события в журнале перед выполнением операций, предусмотренных для этого события по умолчанию.

---

[1] Показано в видеоролике 15 по адресу: http://www.slideshare.net/brandon.aaron/special-events-beyond-custom-events (оригинальная страница в блоге Брендона Аарона (Brandon Aaron): http://brandonaaron.net/blog/2009/06/17/automating-with-special-events).

Будьте внимательны, замещая существующие события собственными реализациями специализированных событий, потому что ваш новый внутренний обработчик будет вызываться для всех событий этого типа, и вы можете по неосторожности внести нежелательные побочные эффекты.

Можно, например, расширить событие `click`, добавив в него поддержку щелчка правой кнопкой мыши, как описывается в следующем разделе. В этом примере также выполняется слежение за событием `contextmenu`, но на этот раз оно преобразуется в обычное событие `click` и передачей в `event.which` информации о том, какой кнопкой выполнен щелчок.

13.3.1. Добавление поддержки правой кнопки в событие click

Было бы гораздо удобнее использовать существующее событие `click`, поддерживающее событие щелчка правой кнопкой мыши, а не создавать отдельное событие `rightclick`. Событие `click` предполагает такую возможность, поставляя вместе с событием объект, атрибут `which` которого идентифицирует кнопку мыши. Но большинство браузеров перехватывают эти события еще до того, как они попадут в сценарий JavaScript.

Отдельное событие `rightclick` можно переделать в расширение стандартного события `click`. При использовании этого расширения специализированного события можно было бы подключать обработчик к событию `click`, как обычно, и анализировать в нем атрибут `event.which`, чтобы определить, какая клавиша была нажата.

```
$('#myDiv').click(function(event) {
    alert('Clicked ' + (event.which == 3 ? 'right' : 'left') +
        ' mouse button');
});
```

В листинге 13.7 приводится расширенная реализация события click.

Листинг 13.7. Добавление поддержки правой кнопки в событие click

```
/* Добавление поддержки правой кнопки в событие click. */
// ❶ Переопределить обработку события click
$.event.special.click = {

    /* Добавляет обработчик щелчка правой кнопкой.
       @param data (object, optional) любые данные,
                                      переданные в вызов bind
       @param namespaces (string[]) любые пространства имен,
                                    переданные в вызов bind */
```

```
    // ❷ Выполняет инициализацию при подключении первого обработчика
    setup: function(data, namespaces) {
        // ❸ Подключить внутренний обработчик
        $(this).addClass('right-clickable').
            bind('contextmenu', $.event.special.click.handler);
        // ❹ Подключить обработчик по умолчанию
        return false;
    },

    /* Удаляет обработчик события щелчка правой кнопкой мыши.
       @param namespaces (string[]) любые пространства имен,
                                    переданные в вызов unbind */
    // ❺ Выполняет заключительные операции
    teardown: function(namespaces) {
        // ❻ Отключить внутренний обработчик
        $(this).removeClass('right-clickable').
            unbind('contextmenu', $.event.special.click.handler);
        // ❼ Отключить обработчик по умолчанию
        return false;
    },

    /* Реализует фактическую обработку события.
       @param event (Event) объект с информацией о событии
       @return (boolean) false - чтобы подавить поведение по умолчанию */
    // ❽ Внутренний обработчик события
    handler: function(event) {
        event.type = 'click';
        event.which = 3;
        // ❾ Изменить настройки и вызвать пользовательскую
              функцию-обработчик
        return $.event.dispatch.apply(this, arguments);
    }
};
```

На этот раз объявлением специализированного события переопределяется обработка стандартного события `click` ❶. Новая функция `setup` ❷ добавляет еще один класс CSS в элемент для нужд оформления и связывает внутренний обработчик с событием `contextmenu`, чтобы получить возможность обрабатывать событие щелчка правой кнопкой ❸. Функция `setup` в данном случае *должна* возвращать `false` ❹, чтобы сообщить инфраструктуре поддержки событий в библиотеке jQuery, что она должна продолжить обычную обработку этого события – подключить свой, стандартный обработчик события `click`. Этот обработчик будет обслуживать обычные события `click`, а ваш обработчик будет заниматься только событием щелчка правой кнопкой.

Как и прежде, функция `teardown` ❺ должна отменить все изменения, произведенные в функции `setup`, – удалить класс CSS и отключиться

от события `contextmenu` ❻. И снова эта функция *должна* вернуть `false` ❼, чтобы jQuery смогла отключить стандартный обработчик события `click` для данного элемента.

Внутренний обработчик ❽ будет вызываться по событию `contextmenu`. Это событие необходимо преобразовать в событие `click` и установить атрибут `event.which`, чтобы показать, что щелчок выполнен правой кнопкой (в документации описывается, что с этой целью должно использоваться значение 3: http://api.jquery.com/event.which/). Затем обработка события продолжается как обычно (с использованием измененного объекта события), вызовом встроенной функции `$.event.dispatch` ❾.

13.4. Другие функциональные возможности событий

Инфраструктура специализированных событий также предоставляет дополнительные функциональные возможности, которые можно использовать при обработке собственных событий. Например, можно определить реакцию по умолчанию на событие, подавить всплытие события и добавить функции для вызова до и после стандартной обработки события. Обо всех этих возможностях рассказывается в следующих разделах.

13.4.1. Реакция по умолчанию на события

Действие, предусмотренное по умолчанию, выполняется после того, как событие завершит всплытие вверх по дереву DOM и будут вызваны все обработчики, встретившиеся на его пути. Примером такого действия по умолчанию может служить загрузка содержимого после щелчка на якорном теге с атрибутом `href`. Любой обработчик события может предотвратить выполнение действия по умолчанию вызовом метода `preventDefault` переданного ему объекта события.

Определить реакцию по умолчанию для своего специализированного события можно, добавив функцию `_default` в объект-обработчик, который, например, будет просто уничтожать событие, как показано в листинге 13.8.

Листинг 13.8. Добавление реакции по умолчанию

```
$.event.special.destroy = {
    ...

    // ❶ Обработчик, реализующий реакцию по умолчанию
```

```
    _default: function(event) {
        $(event.target).remove();
    }
};
```

Внутри функции _default ❶ переменная this будет ссылаться на объект document, потому что к этому моменту событие всплыло до самого верхнего уровня в дереве DOM. Аргумент event содержит всю необходимую информацию о событии. Его атрибут target ссылается на элемент, в котором возникло это событие, а атрибут handleObj обеспечивает доступ тем же настройкам, что и в функциях setup и teardown.

13.4.2. Функции обратного вызова preDispatch и postDispatch

Существует возможность реализовать дополнительную обработку специализированного события, перед тем как оно будет передано какому-либо обработчику, добавив функцию preDispatch в объект-обработчик. Эта функция вызывается в ходе стандартной диспетчеризации события, и с ее помощью можно прервать обработку события, вернув значение false.

Поддержка функции preDispatch была добавлена в версии jQuery 1.7.2.

Например, запрет обработки события rightclick можно было бы поместить в функцию preDispatch, а не во внутренний обработчик, как было показано выше. Внутри этой функции переменная this ссылается на текущий элемент, а аргумент event хранит обычную информацию о событии. В листинге 13.9 представлена одна из возможных реализаций этой функции.

Листинг 13.9. Добавление функции preDispatch

```
$.event.special.rightclick = {
    ...

    // ❶ Внутренний только преобразует событие
    handler: function(event) {
        event.type = $.event.special.rightclick.eventType;
        return $.event.dispatch.apply(this, arguments);
    },

    // ❷ Дополнительная обработка осуществляется в функции preDispatch
    preDispatch: function(event) {
        return !($.event.rightclickDisabled.length &&
            $(this).is($.event.rightclickDisabled));
    }
};
```

Код, реализующий запрет на обработку события, можно убрать из внутреннего обработчика handler ❶ и перенести в функцию preDispatch ❷. Эта функция возвращает false, если текущий элемент присутствует в списке элементов, для которых запрещено генерировать событие ❸.

Аналогично можно реализовать дополнительную обработку события, которая будет выполняться в конце стандартной процедуры диспетчеризации, добавив функцию postDispatch. Эта функция будет вызываться после передачи события всем обработчикам, и в ней отсутствует возможность изменить нормальный порядок его обработки.

Поддержка функции postDispatch была добавлена в версии jQuery 1.7.2.

Например, с помощью этой функции можно было бы реализовать журналирование событий, как показано в листинге 13.10. Внутри данной функции переменная this ссылается на текущий элемент, а аргумент event хранит обычную информацию о событии.

Листинг 13.10. Добавление функции postDispatch

```
$.event.special.rightclick = {
    ...

    // ❶ Дополнительная обработка осуществляется в функции postDispatch
    postDispatch: function(event) {
        console.log(event.type + ' on ' + event.target.nodeName);
    }
};
```

Функция postDispatch ❶ осуществляет дополнительную обработку события после обработки его всеми имеющимися обработчиками. В данном случае в журнал выводится сообщение с информацией о событии: тип события и имя элемента, в котором оно возникло.

13.4.3. Предотвращение всплытия события

Специализированные события также поддерживают возможность предотвратить всплытие за счет установки атрибута noBubble объекта-обработчика события в значение true. jQuery использует эту особенность, чтобы предотвратить всплытие событий load к объекту window.

```
$.event.special.load = {
    // Предотвратить всплытие событий image.load до обработчика window.load
    noBubble: true
};
```

Атрибут noBubble был добавлен в версии jQuery 1.7.1.

13.4.4. Автоматическое связывание и делегирование

Атрибуты `bindType` и `delegateType` объекта-обработчика события позволяют указать типы событий для привязки и делегирования специализированного события, и библиотека jQuery будет автоматически перехватывать эти события и вызывать пользовательские обработчики данного события. Например, jQuery использует этот механизм для отображения специализированных событий `mouseenter` и `mouseleave` непосредственно во встроенные события `mouseover` и `mouseout`.

Для дальнейшей автоматизации подобного процесса инфраструктура событий отыскивает функцию `handle` внутри объекта-обработчика и вызывает ее в ответ на отображаемые события. Вы должны написать эту функцию, если требуется выполнить какие-то дополнительные операции, такие как, например, изменение типа события. В противном случае пользовательская функция-обработчик будет вызываться непосредственно.

Атрибуты `bindType` и `delegateType` были добавлены в версии jQuery 1.7.1.

Но при использовании такого подхода функции `setup` и `add` больше не вызываются, так как предполагается, что прямое отображение во встроенные события устраняет потребность в этих вызовах. Как следствие функции `teardown` и `remove` становятся ненужными (хотя иногда их все еще вызывают по каким-либо причинам).

Например, объект-обработчик события `rightclick`, рассматривавшийся в предыдущих разделах, можно переписать с использованием автоматического связывания и делегирования, как показано в листинге 3.11.

Листинг 3.11. Использование автоматического связывания и делегирования

```
// ❶ Определение события щелчка правой кнопкой мыши
$.event.special.rightclick = {

    // ❷ Отображение во встроенное событие
    bindType: 'contextmenu',
    delegateType: 'contextmenu',
    eventType: 'rightclick',

    // ❸ Обработчик, автоматически вызываемый в ответ
    //    на встроенное событие
    handle: function(event) {
        event.type = $.event.special.rightclick.eventType;
```

```
        // ❹ Вызвать пользовательскую функцию-обработчик
        return event.handleObj.handler.apply(this, arguments);
    }
};
```

Как и прежде, объект-обработчик события определяется под именем `rightclick` ❶, но на этот раз он содержит альтернативную реализацию. Здесь используются атрибуты `bindType` и `delegateType` ❷ для отображения данного события в существующее событие `contextmenu`. Функция `handle` ❸ вызывается для обработки этого события и изменяет тип события перед вызовом пользовательской функции-обработчика ❹, доступной через объект `event.handleObj`.

Обратите внимание, что эта реализация не добавляет класс CSS к целевым элементам. Вам также может потребоваться включить в функцию `handle` обслуживание запрета на обработку события из функции `handler` в первоначальной реализации события `rightclick`.

Полный код реализации описанных здесь специализированных событий досупен для загрузки на веб-сайте книги.

Это нужно знать

Создавайте новые события, когда требуется обеспечить единообразие обработки каких-либо ситуаций в разных браузерах.
В библиотеке jQuery имеется инфраструктура поддержки специализированных событий, предназначенная для обработки событий стандартным образом.
Добавление новых или переопределение существующих событий осуществляется через точку интеграции `$.event.special`.
Объект-обработчик вашего события может следить за появлением других событий, изменять их или взамен возбуждать другие события.
Функции `setup` и `teardown` внутри объекта-обработчика вызываются только один раз для каждого элемента, тогда как функции `add` и `remove` вызываются для каждого подключаемого или отключаемого обработчика событий.
Если вернуть `false` из функций `setup` и `teardown`, jQuery продолжит стандартную процедуру настройки обработки события.

Попробуйте сами

Создайте событие `reset` для формы, чтобы с его помощью запрашивать подтверждение у пользователя, прежде чем выполнить фактический сброс всех элементов формы в исходное состояние. Для форм уже поддерживается стандартное событие `reset`, поэтому вам потребуется позволить библиотеке jQuery обрабатывать его как обычно. Выполните отмену события `reset` вызовом функции `event.preventDefault`.

13.5. В заключение

Библиотека jQuery упрощает взаимодействия со встроенными событиями JavaScript, предоставляя несколько способов подключения обработчиков определенных событий к указанному элементу. Она также поддерживает делегирование событий, чтобы дать возможность создавать единственный обработчик для множества вложенных элементов или для обработки событий в элементах, которые пока отсутствуют в дереве DOM.

Для интеграции новых событий в стандартную процедуру обработки в библиотеке jQuery имеется собственная инфраструктура поддержки событий. С ее помощью можно зарегистрировать новое событие, определив объект-обработчик, знающий, как инициализировать их обработку, какие заключительные операции при необходимости требуется выполнить и как передавать эти события подключенным пользовательским функциям-обработчикам.

В этой главе было показано, как с помощью инфраструктуры специализированных событий преобразовать встроенное событие `contextmenu` в более понятное событие `rightclick`. Затем была продемонстрирована расширенная версия этого события – событие `rightmulticlick`, объект-обработчик которого следит за количеством щелчков правой кнопкой мыши для каждого элемента в отдельности. И в заключение вы увидели, как можно изменить стандартное событие `click`, включив в него поддержку правой кнопки мыши.

В заключительной главе мы поближе познакомимся с расширением Validation и с возможностью его увеличения собственными правилами проверки.

Глава 14

Создание правил проверки

Эта глава охватывает следующие темы:

- расширение Validation;
- добавление собственных правил проверки.

Несмотря на то что оно не является частью библиотеки jQuery, расширение Validation, созданное Йерном Заффererом (Jörn Zaefferer), получило весьма широкое распространение и имеет собственные точки интеграции (http://jqueryvalidation.org). Это расширение помогает организовать проверку введенных пользователем данных на стороне клиента и избежать ненужных запросов, которые пришлось бы выполнять для исправления ошибок, допущенных пользователем, таких как отсутствие данных в обязательных полях или неправильно отформатированный адрес электронной почты.

В числе правил, входящих в состав расширения, можно назвать: `required` – требует обязательного заполнения поля, `digits` или `number` – для числовых полей ввода, `min` и `max` – определяют минимальное и максимальное значения, `email` и `url` – для полей ввода адресов электронной почты и URL, `equalTo` – для сравнения значений полей. Эти правила можно объединять в одном поле, создавая сложные правила проверки, такие как, например, для обязательных числовых полей с максимально возможным значением.

Расширение управляет применением этих правил в соответствующие моменты времени – например, когда происходит событие ввода в поле или когда выполняется попытка отправить форму на сервер, – и отображением соответствующих сообщений об ошибках, если это необходимо. Цель расширения – предоставить ненавязчивый способ проверки полей, отображения сообщений только после действий пользователя и удалять их сразу же, как только они станут неактуальны. Размещение сообщений об ошибках и их внешний вид при

необходимости могут контролироваться пользователем, хотя чаще принято использовать настройки по умолчанию.

Если встроенных правил оказывается недостаточно, всегда можно определить свои правила и интегрировать их в расширение. В простейшем представлении правило – это функция, возвращающая `true`, если значение поля допустимо, и `false` – если нет.

В этой главе вы увидите, как определять собственные правила, проверяющие введенный текст, сопоставляя его с шаблонами, и устанавливающие условия, которые распространяются на несколько полей, а также как применять их в своих приложениях.

14.1. Расширение Validation

Расширение Validation применяется к формам и обеспечивает возможность проверки полей этих форм перед отправкой на сервер (как показано на рис. 14.1). Производя такие проверки на стороне клиента,

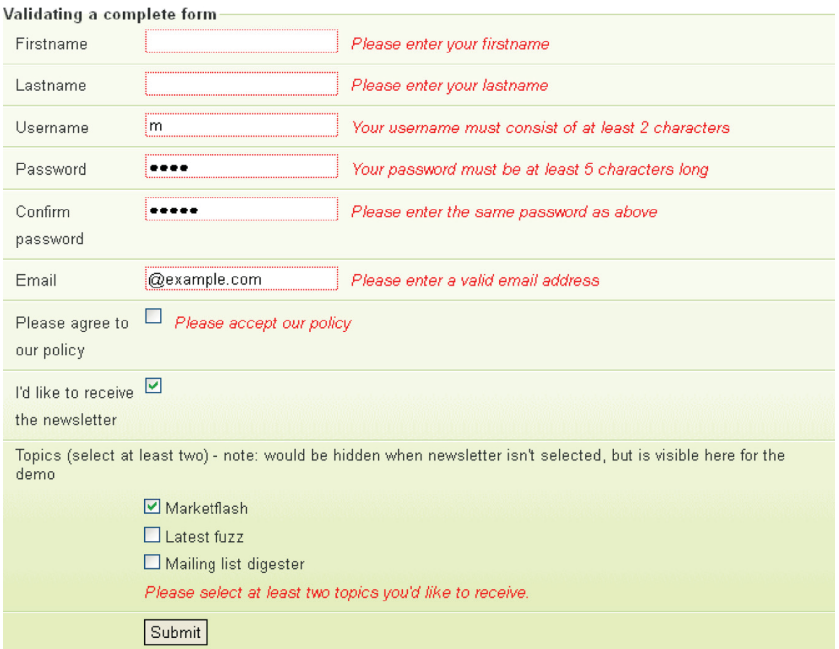


Рис. 14.1 ❖ Расширение Validation вывело несколько сообщений об ошибках, выявленных при проверке полей

можно предотвратить выполнение лишних сетевых операций, только чтобы получить сообщение об ошибке в возвращаемой странице. Обратите внимание, что при создании своих веб-сайтов никогда не следует полагаться на проверку, выполняемую на стороне клиента. Те же самые правила проверки должны применяться на стороне сервера, прежде чем пустить в обработку принятую информацию.

В этом разделе рассказывается, как расширение Validation назначает правила определенным полям в веб-странице – либо через метаданные, подключаемые к каждому HTML-элементу, либо посредством параметров, передаваемых в вызов функции инициализации расширения.

14.1.1. Назначение правил проверки

Это расширение позволяет назначать правила проверки для полей ввода несколькими разными способами, обеспечивая тем самым определенную гибкость в реализации страницы. Двумя основными способами являются: назначение через классы и атрибуты полей (метаданные) и посредством параметров настройки на этапе инициализации расширения.

Назначение правил через метаданные элемента

В первом способе в поля ввода добавляются классы, соответствующие требуемым правилам, для которых не нужно указывать дополнительные параметры. Если возникнет необходимость передать значение для правила, такое правило должно быть определено как атрибут с именем, совпадающим с именем правила, и значением необходимого параметра. Потом, когда расширение будет инициализировано для формы в целом, эти классы и атрибуты будут преобразованы в соответствующие им правила.

Например, в листинге 14.1 показано, как определять правила с помощью встраиваемых атрибутов.

Листинг 14.1. Применение встроенных правил

```
<script type="text/javascript">
$(function() {
    // ❶ Выполнить проверку всей формы
    $('#myform').validate();
});
</script>
...
<form id="myform" method="get">
```

```
    <!-- ❷ Поле ввода имени обязательно для заполнения -->
    <input type="text" name="firstName" class="required">
    <!-- ❸ Возраст должен быть числом не менее 18 -->
    <input type="text" name="age" class="digits" min="18">
    <input type="submit" value="Submit">
</form>
```

Здесь выполняется применение расширения Validation к форме в целом ❶. В результате расширение производит сканирование атрибутов полей и применяет указанные правила. Присутствие класса `required` в поле `firstName` делает его обязательным для заполнения ❷. Поле `age` является необязательным, но если в нем присутствует какое-либо значение, оно должно быть числом (целым), согласно классу `digits` ❸. Атрибут `min` указывает, что значение этого поля должно быть не меньше значения параметра, равного 18.

Назначение правил через параметры

Чтобы определить правила проверки в процессе инициализации, следует указать аргумент с атрибутом `rules` в вызове функции расширения. Этот аргумент должен быть объектом с атрибутами для каждого поля и соответствующими правилами в этих атрибутах.

В листинге 14.2 представлена та же самая форма, что и в листинге 14.1, но с определениями правил в вызове функции инициализации расширения.

Листинг 14.2. Применение правил проверки в вызове функции инициализации

```
<script type="text/javascript">
$(function() {
    // ❶ Выполнить проверку всей формы
    $('#myform').validate({
        rules: {
            // ❷ Поле ввода имени обязательно для заполнения
            firstName: 'required',
            // ❸ Возраст должен быть числом не менее 18
            age: {
                digits: true,
                min: 18
            }
        }
    });
});
</script>
...
<form id="myform" method="get">
```

```
    <!-- ❹ Поля ввода не имеют дополнительных атрибутов -->
    <input type="text" name="firstName">
    <input type="text" name="age">
    <input type="submit" value="Submit">
</form>
```

И снова здесь функция расширения Validation вызывается для формы в целом ❶. Но на этот раз применяемые правила проверки определяются атрибутом `rules` аргумента. Атрибуты самого объекта `rules` определяют имена целевых полей (обратите внимание, что идентификация элементов выполняется по атрибуту `name`, а не `id`). Значением каждого атрибута должно быть имя правила, если применяется единственное правило (не требующее дополнительных параметров, как, например, правило `required`). Когда применяется несколько правил или когда правило требует наличия параметра (как, например, правило `min`), значением атрибута должен быть другой объект с атрибутами, соответствующими именам правил и значениям параметров. Если правило не имеет параметра, в качестве значения передается `true`. Как и в предыдущем примере, здесь поле `firstName` объявляется обязательным ❷, а поле `age` должно содержать только цифры, причем его значение не должно быть меньше 18 ❸. Сами поля формы в разметке HTML не имеют никаких дополнительных классов или атрибутов ❹. Такой подход позволяет отделить представление формы от ее реализации.

> Если имя поля содержит неалфавитно-цифровые символы, его следует заключить в одиночные кавычки при использовании в качестве имени атрибута в определении правил. Например: `rules: {'first-name': 'required'}`.

14.2. Добавление новых правил проверки

Несмотря на большое количество правил, уже входящих в состав расширения, иногда бывает необходимо применить какое-то более специфическое правило, как, например, проверка формата номера социального обеспечения США (Social Security Number, SSN). Вызовом функции `addMethod` объекта-валидатора можно определить свое собственное правило, которое может применяться наравне со встроенными. Ваше правило будет добавлено в список доступных правил по имени, указанному в определении. После этого вы сможете использовать это имя и назначать правило для проверки полей формы.

Часто бывает необходимо проверить введенные данные на соответствие некоторому шаблону, такому как, например, телефонный номер. В действительности в расширении Validation имеется файл

additional-methods.js, содержащий такого рода правила, включая правило pattern сопоставления с пользовательским шаблоном и правило phoneUS проверки на соответствие шаблону телефонных номеров в США.

Далее мы реализуем собственную версию правила проверки соответствия шаблону, чтобы увидеть, как это делается, а затем создадим генератор правил подобного типа, чтобы упростить их определение в будущем.

14.2.1. Добавление правила проверки соответствия шаблону

Для проверки значений на соответствие некоторому шаблону в JavaScript необходимо использовать объекты регулярных выражений, встроенные в язык (примеры использования регулярных выражений в JavaScript приводятся в приложении А). Регулярные выражения определяются в виде строк символов, которые должны совпадать буквально или имеют некоторый особый смысл в регулярных выражениях, как, например, оператор выбора из нескольких альтернатив, квантификаторы, описывающие возможное число повторений, или представления классов символов.

Чтобы создать правило, выполняющее сопоставление с шаблоном и пригодное для использования в разных ситуациях, необходимо обеспечить возможность передачи шаблона в качестве параметра для инициализации правила. Например, можно организовать проверку поля ввода на соответствие шаблону номера социального обеспечения США, как показано ниже:

```
$('#myform').validate({rules: {
    ssn: {matches: /^\d{3}-\d{2}-\d{4}$/}
}});
```

Этот шаблон требует, чтобы поле начиналось (^) с трех цифр (\d{3}), за которыми должны следовать дефис (-), две цифры (\d{2}), еще один дефис (-), и завершаться ($) четырьмя цифрами (\d{4}). Для большей гибкости шаблон должен передаваться правилу в виде строки, а не объекта RegExp, как показано в этом примере. Обратите внимание на необходимость экранирования обратных слэшей (\) и одиночных кавычек (') при передаче шаблона в виде строки, вследствие чего шаблон из примера выше примет вид: '^\\d{3}-\\d{2}-\\d{4}$'.

В листинге 14.3 показано, как реализовать правило, выполняющее сопоставление с шаблоном. Как и в случае с другими расширения-

ми в предыдущих главах, код этого расширения заключен в вызов анонимной функции, чтобы скрыть внутренние переменные в новой области видимости и гарантировать возможность ссылки на объект jQuery через идентификатор $.

Листинг 14.3. Добавление правила проверки соответствия шаблону

```
/* Правило для проверки на соответствие регулярному выражению.
   @param value   (string) текущее значение поля
   @param element (jQuery) текущее поле
   @param param   (string или RegExp) шаблон для сопоставления
   @return (boolean) true - если соответствует, false - если нет */
// ❶ Определение нового правила
$.validator.addMethod('matches', function(value, element, param) {
        // ❷ Создать объект регулярного выражения
        var re = param instanceof RegExp ? param : new RegExp(param);
        // ❸ Проверить значение
        return this.optional(element) || re.test(value);
    },
    // ❹ Формат сообщения об ошибке
    $.validator.format('Please match this format "{0}".'));
```

Чтобы определить новое правило, необходимо вызвать функцию $.validator.addMethod ❶. В качестве аргументов этой функции передаются: имя нового правила (чтобы его можно было назначать отдельным полям), функция, выполняющая проверку элемента и его значения, а также сообщение об ошибке, которое должно быть отображено, если поле не пройдет проверку.

Функция, выполняющая проверку, также принимает три аргумента: текущее значение элемента для проверки (value), ссылку на этот элемент DOM (element) и параметры, переданные при назначении правила на этапе инициализации (param).

Проверка начинается с создания объекта регулярного выражения ❷. Если в аргументе передан готовый объект, для проверки будет использоваться этот объект. В противном случае на основе переданного шаблона создается новый объект RegExp.

Функция, выполняющая проверку, должна вернуть true, если поле и его значение соответствуют правилу, и false, если нет ❸. На случай, если поле является необязательным, следует допустить возможность отправки пустого поля вызовом стандартной функции optional, которой нужно передать ссылку на элемент. В противном случае (если поле содержит некоторое значение) необходимо применить регулярное выражение к текущему значению, вызвав функцию test объекта регулярного выражения.

Если значение не соответствует шаблону, на странице будет выведено сообщение об ошибке, переданное в вызов функции `addMethod` ❹. Сообщение может быть представлено простой статической строкой, но при желании в текст сообщения можно включать динамические значения из параметров, вызвав функцию `$.validator.format`. Она принимает строку сообщения с параметрами и возвращает новую функцию, которая должна вызываться для создания окончательного текста сообщения. Динамические значения (параметры) передаются в строке сообщения в виде последовательности `{n}`, где `n` – индекс соответствующего элемента в массиве параметров или 0, если в сообщении имеется только один параметр.

В это правило можно передавать шаблон, который будет использоваться для сопоставления. Его можно также объединять с другими правилами, для организации более сложной проверки. На рис. 14.2 показан результат применения правила, выполняющего сопоставление с шаблоном для проверки соответствия ввода формату номера социального страхования, с использованием определения из листинга 14.4.

Рис. 14.2 ❖ Результат применения правила, выполняющего сопоставление с шаблоном

Листинг 14.4. Пример использования правила, выполняющего сопоставление с шаблоном

```
$('#myform').validate({
    rules: {
        ssn1: {
            required: true,
            // ❶ Сопоставление со строковым шаблоном
            matches: '^\\d{3}-\\d{2}-\\d{4}$'
        },
        ssn2: {
            // ❷ Сопоставление с объектом регулярного выражения
            matches: /^\d{3}-\d{2}-\d{4}$/
        }
    },
    messages: {
```

```
        ssn2: {
            // ❸ Собственное сообщение об ошибке
            matches: 'Please enter a valid SSN'
        }
    }
});
```

В определении первого правила проверки для поля ввода номера социального страхования передается шаблон в виде строки ❶. Это поле также объявляется обязательным к заполнению применением к нему правила `required`. В определении правила проверки для второго поля шаблон передается в виде готового объекта регулярного выражения ❷, и это поле является необязательным. Для отдельных правил и полей можно также переопределять сообщения об ошибках, чтобы сообщить пользователю более осмысленную информацию ❸. Внутри атрибута `messages` следует идентифицировать поле по имени и правило, указанное в атрибуте `rules`, а также текст сообщения для этого правила.

Если потребуется определить правила проверки в разметке HTML, добавьте в него соответствующие классы и атрибуты, как показано ниже, где объявляется поле, обязательное для заполнения, с проверкой ввода на соответствие шаблону.

```
<input type="text" id="ssn1" name="ssn1"
    class="required" matches="^\d{3}-\d{2}-\d{4}$">
```

Итак, вы узнали, как применять правило `matches` к полям для проверки соответствия ввода произвольному регулярному выражению. Такие регулярные выражения могут быть весьма сложными для понимания и сопровождения, поэтому в следующем разделе мы посмотрим, как обеспечить необходимую функциональность правила сопоставления, описывая его назначение.

14.2.2. Генерирование правил сопоставления с шаблоном

Функция, выполняющая проверку путем сопоставления с регулярным выражением, представленная в предыдущем разделе и повторяющаяся в листинге 14.5, принимает регулярное выражение в качестве аргумента (`param`) от инфраструктуры поддержки правил. Однако точно так же можно было бы указать шаблон при создании правила и получить специализированное правило, более простое в понимании и применении, как показано на рис. 14.3.

Рис. 14.3 ❖ Сообщение об ошибке, полученное от генератора правил сопоставления с шаблоном

Листинг 14.5. Предыдущее правило, реализующее сопоставление с шаблоном

```
function(value, element, param) {
    var re = param instanceof RegExp ? param : new RegExp(param);
    return this.optional(element) || re.test(value);
},
```

В листинге 14.6 показано, как реализовать функцию, генерирующую аналогичные функции сопоставления с шаблонами для использования в определениях правил.

Листинг 14.6. Генератор правил, выполняющих сопоставление с шаблоном

```
/* Содает правило проверки для указанного регулярного выражения.
   @param pattern (string или RegExp) шаблон для сопоставления
   @return (function) функция для использования в определении правила */
// ❶ Определение функции-генератора
function createRegExpRule(pattern) {
    // ❷ Создать объект регулярного выражения
    var re = pattern instanceof RegExp ? pattern : new RegExp(pattern);
    // ❸ Вернуть функцию для использовании в определении правила
    return function(value, element, param) {
        // ❹ Правило проверки
        return this.optional(element) || re.test(value);
    };
}
```

Здесь определяется функция `createRegExpRule`, генерирующая новые функции для правил, выполняющие проверки ❶. Она принимает один аргумент – шаблон для сопоставления (`pattern`) – и возвращает функцию, которую можно передать расширению Validation при определении нового правила проверки.

Как и прежде, необходимо на основе полученного шаблона создать объект регулярного выражения для выполнения фактической проверки ❷ либо использовать имеющийся объект. Это можно сделать до возврата функции, выполняющей проверку, и избежать необходимости повторно выполнять эти действия при каждом применении правила.

Затем возвращается фактическая функция, выполняющая поверку ❸. Хотя эта функция принимает три аргумента, последний из них (param) больше не используется и игнорируется. Функция повторяет последнюю строку универсальной функции проверки из предыдущего раздела, проверяя обязательность поля перед сопоставлением текущего значения с шаблоном, полученным ранее ❹. Операция сопоставления возвращает true, если значение соответствует шаблону, и false , если нет.

Функцию-генератор легко использовать для создания правил, выполняющих сопоставление с шаблоном, как показано в листинге 14.7.

Листинг 14.7. Использование генератора правил на основе регулярных выражений

```
/* Правило, выполняющее проверку соответствия ввода формату
   номеров социального страхования в США.
   @return (boolean) true - если соответствует, false - если нет */
// ❶ Определение нового правила
$.validator.addMethod('ssn',
    // ❷ Создать функцию, выполняющую проверку
    createRegExpRule('^\\d{3}-\\d{2}-\\d{4}$'),
    // ❸ Определить сообщение об ошибке
    'Please enter a SSN - nnn-nn-nnnn.');
```

Как и прежде, для регистрации нового правила вызывается функция $.validator.addMethod ❶. Ей передаются: имя правила, идентифицирующее используемый шаблон, функция, фактически выполняющая проверку сопоставлением с шаблоном, возвращаемая вызовом функции createRegExpRule ❷, и сообщение об ошибке ❸ в виде простой строки, потому что в данном случае никаких параметров передавать не требуется.

Чтобы применить новое правило к конкретному полю, больше не нужно указывать какие-либо параметры, и правило может быть идентифицировано исключительно по имени.

```
$('#myform').validate({rules: {
    ssn3: 'ssn'
}});
```

Если требуется применить несколько правил к одному полю, их нужно перечислить в объекте и использовать значение true в качестве значения правила в определении. Например, чтобы дополнительно объявить поле обязательным к заполнению, можно использовать следующий код:

```
$('#myform').validate({rules: {
    ssn3: {required: true, ssn: true}
}});
```

Только что представленные правила применяются к единственному полю и его значению, однако есть возможность создавать правила, выполняющие проверку нескольких полей, как описывается в следующем разделе.

14.3. Добавление правила для проверки нескольких полей

Правила проверки не обязательно должны применяться к единственному полю ввода и могут оказывать влияние на целые группы взаимосвязанных полей (таких как поля ввода дня, месяца и года, образующие единую дату), поскольку значения полей по отдельности могут быть вполне допустимыми, а в группе – нет.

Расширение Validation уже имеет правило `equalTo`, позволяющее проверять равенство значений двух полей, когда требуется подтвердить ввод пароля или адрес электронной почты. Кроме того, в правила можно добавлять зависимости, чтобы они применялись только в определенных ситуациях. Например, ниже показано, как потребовать обязательность поля, если помечено другое поле:

```
$('#myform').validate({rules: {
    myField: {required: '#otherField:checked'}
}});
```

Если подобных встроенных возможностей окажется недостаточно, можно определить собственные правила проверки взаимосвязанных полей. Допустим, что в некотором голосовании пользователь может распределить определенное число голосов между несколькими пунктами в списке, чтобы показать значимость каждого пункта, а вам требуется проверить, чтобы общая сумма голосов совпадала с требуемым числом. Проверка отдельного поля не позволит установить допустимость общей суммы. Для решения этой задачи можно создать новое правило, но сначала требуется предусмотреть возможность группировки полей, чтобы отображалось только одно сообщение об ошибке.

14.3.1. Группировка полей

При инициализации расширения Validation в качестве параметров можно передавать определения групп взаимосвязанных полей. Для каждой группы должны указываться ее имя и список имен полей, входящих в нее (перечисленных через пробел). Обратите внимание,

что в качестве имен следует использовать значения атрибута `name`, а не атрибута `id`.

```
$('#myform').validate({groups: {
    address: 'address1 address2 city state postcode'
}, ...});
```

Для каждой группы будет генерироваться единственное сообщение об ошибке, поэтому для всей группы нужно определить единственный параметр `errorPlacement`.

14.3.2. Определение правила для группы полей

Вернемся к необходимости проверки распределения голосов между несколькими полями в форме опроса. Нам необходимо определить правило, которое найдет сумму всех голосов в полях, составляющих единую группу, и сравнит ее с допустимым числом. Результат применения этого правила показан на рис. 14.4, где также видно, что для всей группы отображается единственное сообщение об ошибке, если сумма значений полей окажется недопустимой. Обратите внимание, что в сообщении указывается общее ожидаемое число голосов.

Для большей гибкости и возможности многократного использования правила ожидаемая сумма и способ выбора взаимосвязанных

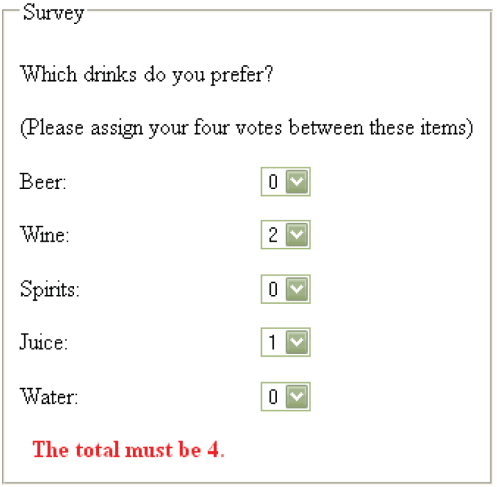


Рис. 14.4 ❖ Правило проверки общей суммы в действии: если сумма значений полей оказывается недопустимой, выводится одно общее сообщение об ошибке

полей должны указываться в виде параметров правила. Реализация данного правила приводится в листинге 14.8.

Листинг 14.8. Определение правила для группы полей

```
/* Правило, проверяющее общую сумму значений группы полей.
   @param value   (string) значение текущего поля
   @param element (jQuery) текущее поле
   @param param   (number и string) требуемая сумма
                                    и селектор для выбора всех полей
   @return (boolean) true - если значение допустимо, false - если нет */
// ❶ Определение нового правила
$.validator.addMethod('totals', function(value, element, param) {
        var sum = 0;
        // ❷ Сумма значений всех взаимосвязанных полей
        $(param[1]).each(function() {
            sum += parseInt($(this).val(), 10);
        });
        // ❸ Сравнить полученную сумму с ожидаемой
        return sum == param[0];
    },
    // ❹ Текст сообщения об ошибке
    $.validator.format('The total must be {0}.'));
```

Регистрация нового правила с именем `totals` производится вызовом функции `addMethod` ❶. Как и прежде, функция, выполняющая проверку, принимает в качестве аргументов значение текущего поля (`value`), само поле (`element`) и параметры настройки правила (`param`). Данная функция использует только последний аргумент, потому что в подобном случае интерес представляет группа полей в целом, а не отдельное поле.

После инициализации переменной `sum` функция выбирает все взаимосвязанные поля (используя селектор из второго элемента в массиве параметров `param`) и обрабатывает их по очереди ❷. Из каждого поля извлекается его значение, преобразуется в число (`parseInt`) и прибавляется к текущей сумме (`sum`). После вычисления суммы всех значений она сравнивается с ожидаемой суммой (первый элемент в массиве параметров `param`), и результат сравнения возвращается как результат проверки ❸.

Если сумма значений полей окажется недопустимой, расширение выведет указанное сообщение об ошибке ❹. Чтобы увеличить информативность сообщения, в него включается ожидаемая сумма вызовом функции `$.validator.format` и обозначением места, куда должно быть вставлено число с помощью конструкции `{0}`.

Чтобы задействовать это правило в своей странице, необходимо инициализировать расширение Validation, передав ему свою форму и определив необходимые настройки, как показано в листинге 14.9.

Листинг 14.9. Применение правила для группы полей

```
// ❶ Определить настройки для правила
var allVotes = {
    totals: [4, 'select.item']
};
$('#myform').validate({
    // ❷ Определить группу полей
    groups: {
        items: 'item1 item2 item3 item4 item5'
    },
    rules: {
        // ❸ Назначить проверку для полей
        item1: allVotes,
        item2: allVotes,
        item3: allVotes,
        item4: allVotes,
        item5: allVotes
    },
    // ❹ Настроить сообщение об ошибке
    errorPlacement: function(error, element) {
        // ❺ Если элемент включен в группу...
        if (element.hasClass('item')) {
            // ❻ ... сместить сообщение об ошибке
            error.appendTo(element.closest('fieldset'));
        }
        else {
            // ❼ Иначе использовать местоположение по умолчанию
            error.insertAfter(element);
        }
    }
});
```

Так как новое правило будет применяться к нескольким полям с идентичными настройками, предпочтительнее определить эти настройки в одном месте и повторно использовать их по мере необходимости. В данном примере объект `allVotes` ❶ идентифицирует новое правило, определяет ожидаемую сумму `totals`, равную 4, и селектор `select.item` для выбора полей в группе. Параметры настройки определяются как элементы массива.

При инициализации расширения Validation ему передается описание группы полей ❷, чтобы для них генерировалось только одно сообщение об ошибке. Затем новое правило применяется к каждому

полю по очереди ❸, с использованием общих настроек, определенных выше. При таком подходе проверка будет выполняться в случае изменения значения любого поля.

С целью получить контроль над выводом сообщения об ошибке для этих полей переопределяется функция `errorPlacement` ❹. Если текущий элемент принадлежит к группе (определяется по наличию общего класса) ❺, для сообщения об ошибке определяется позиция в конце контейнера – ближайшего вмещающего элемента `fieldset` ❻. В противном случае используется позиция по умолчанию ❼ – рядом с целевым полем.

Полный код реализации описанных здесь правил доступен для загрузки на веб-сайте книги.

Это нужно знать

Расширение Validation позволяет определять правила для проверки содержимого полей перед отправкой формы на сервер.
Создавайте новые правила для удовлетворения специфических требований.
Выполняйте регистрацию новых правил вызовом `$.validator.addMethod`.
Правила могут принимать дополнительные параметры настройки.
Сообщения об ошибках в правилах могут включать дополнительные параметры.
Правила могут проверять допустимость не только одного поля.
Используйте параметр `groups` для определения групп взаимосвязанных полей, чтобы обеспечить вывод сообщения об ошибке только для группы в целом.

Попробуйте сами

Создайте правило, требующее наличия значения в поле, но только одного из указанных в массиве допустимых значений.

```
field: {oneof: ['one', 'two', 'three']}
```

Затем повторно реализуйте это правило с использованием генератора на основе регулярного выражения.
Подсказка: используйте символ | для разделения альтернатив.

14.4. В заключение

Расширение Validation является одним из наиболее часто используемых расширений. Оно позволяет упростить реализацию проверки полей форм и вывод сообщений об ошибках. Несмотря на большое количество встроенных правил, включая дополнительные, поставляемые в виде отдельного модуля, иногда их возможностей оказывается недостаточно. К счастью, расширение позволяет добавлять собствен-

ные правила и использовать их наряду со встроенными, предоставляя специальную точку интеграции.

Регистрация собственных правил производится вызовом функции `$.validator.addMethod`, которой передается функция, выполняющая проверку и возвращающая `true`, если проверка пройдена успешно, и `false` в противном случае. Добавив универсальное правило, выполняющее проверку с применением регулярного выражения, вы сможете использовать его во множестве ситуаций, передавая шаблоны, подходящие для решения конкретной задачи. Кроме того, данный подход можно расширить, реализовав на его основе динамическое создание правил, упрощающих реализацию проверки и улучшающих удобочитаемость программного кода.

Расширением Validation поддерживается также возможность создания правил для проверки группы полей и вывода единственного сообщения об ошибке для всей группы.

При создании собственного расширения подумайте, как удобнее было бы использовать и расширять его. Добавляя свои точки интеграции, вы сможете упростить вероятность наращивания возможностей своего расширения другими пользователями и тем самым расширить круг его применения.

Приложение A

Регулярные выражения

Регулярные выражения в JavaScript – это объекты, описывающие шаблоны последовательностей символов. Данные объекты используются для сопоставления строк или частей строк с шаблонами, а также для выполнения операций поиска с заменой. В библиотеке jQuery регулярные выражения используются очень широко для решения таких задач, как парсинг выражений селекторов, определения типа браузера и удаления начальных и конечных пробелов в тексте.

Применение регулярных выражений в JavaScript играет важную роль в повышении эффективности программного кода. Регулярные выражения часто использовались в расширениях, представленных в этой книге, для проверки значений или для разбиения строк на составляющие их компоненты. Любой программист должен знать синтаксис регулярных выражений и уметь применять их на практике.

Ресурсы в Сети

Дополнительную информацию о регулярных выражениях и примеры их применения в JavaScript можно найти в Интернете. Ниже приводится несколько ссылок на различные ресурсы, где можно найти дополнительные ссылки и учебники по этой теме:

- объект JavaScript RegExp – www.w3schools.com/jsref/jsref_obj_regexp.asp[1];
- учебник по регулярным выражениям – https://developer.mozilla.org/en/JavaScript/Guide/Regular_Expressions[2];
- учебник по практическому применению регулярных выражений в JavaScript – www.regular-expressions.info/javascript.html[3];
- еще один учебник по регулярным выражениям – www.learn-javascript-tutorial.com/RegularExpressions.cfm.

---

[1] http://javascript.ru/RegExp – *Прим. перев.*
[2] http://javascript.ru/basic/regular-expression – *Прим. перев.*
[3] http://htmlweb.ru/java/regexp.php – *Прим. перев.*

А.1. Основы регулярных выражений

Создать объект регулярного выражения можно вызовом функции `RegExp`:

```
var re = new RegExp(pattern, modifiers);
```

или используя форму литерала:

```
var re = /pattern/modifiers;
```

Параметр *pattern* в первой версии – это строковое значение, а во второй – литерал регулярного выражения без окружающих его кавычек. Обе формы идентичны, за исключением случаев, когда требуется экранировать служебные символы, добавляя перед ними символ обратного слэша (\). В строковой (первой) версии необходимо экранировать кавычки, используемые для ограничения строки (" или '), и символы обратного слэша, а в литеральной (второй) версии достаточно экранировать только символы слэша (/). Параметр *modifiers* является необязательным и может быть опущен, если дополнительные параметры регулярного выражения не требуются. Они определяются как одна строка в первой версии и как последовательность символов-литералов – во второй. Поддерживаемые модификаторы перечислены в табл. А.1.

Таблица А.1. Модификаторы регулярных выражений

Модификатор	Описание
`i`	Выполняет сопоставление без учета регистра символов
`g`	Поиск всех совпадений (глобальный)
`m`	Символы ^ и $ соответствуют символам перевода строки (многострочный режим поиска)

Параметр *pattern* – это последовательность литералов символов, с которыми выполняется сопоставление, а также метасимволов, позволяющих создавать более сложные шаблоны. В табл. А.2 перечислены типичные шаблоны, а их синтаксис подробно обсуждается в следующем разделе.

Таблица А.2. Примеры типичных регулярных выражений

Назначение	Выражение	Описание
Номера социального страхования в США	`^\d{3}-\d{2}-\d{4}$`	Три цифры, дефис, две цифры, дефис, четыре цифры

Таблица А.2 (окончание)

Назначение	Выражение	Описание
Адрес электронной почты (упрощенный)	`^[\w.]+@[\w.]+\.\w{2,3}$`	Один или более алфавитно-цифровых символов, подчеркиваний или точек; символ `@` (коммерческое *at*); один или более алфавитно-цифровых символов, подчеркиваний или точек; одна точка; два или три алфавитно-цифровых символа или подчеркивания
Дата в формате США	`^(0[1-9]|1[0-2])\/(0[1-9]|[12][0-9]|3[01])\/\d{4}$`	Две цифры (от `01` до `12`), слэш, две цифры (от `01` до `31`), четыре цифры – обратите внимание, что это выражение все еще совпадает с недопустимыми датами, такими как `02/31/2012`

А.2. Синтаксис регулярных выражений

Регулярные выражения конструируются из литералов символов, соответствующих точно самим себе, и последовательностей метасимволов, позволяющих определять более сложные шаблоны. Простые шаблоны можно комбинировать рекурсивно и создавать из них более сложные шаблоны, точно соответствующие потребностям.

> В шаблонах, обсуждаемых в следующем разделе, символы, изображенные наклонным шрифтом, обозначают метки-заполнители, которые должны замещаться текстом, соответствующим для каждого конкретного случая.

Чтобы определить литерал символа, который нельзя ввести непосредственно, можно использовать одну из форм, перечисленных в табл. А.3. Если потребуется обеспечить буквальное сопоставление с любым из метасимволов, его следует экранировать символом обратного слэша (`\`).

Таблица А.3. Литеральные выражения

Шаблон	Описание
`\0`	Совпадает с нулевым символом
`\f`	Совпадает с символом перевода формата
`\n`	Совпадает с символом перевода строки
`\r`	Совпадает с символом возврата каретки

Таблица А.3 (окончание)

Шаблон	Описание
\t	Совпадает с символом табуляции
\v	Совпадает с символом вертикальной табуляции
\cx	Совпадает с управляющим символом *x*
\ooo	Совпадает с символом, имеющим код *ooo* в восьмеричном представлении
\xhh	Совпадает с символом, имеющим код hh в шестнадцатеричном представлении
\unnnn	Совпадает с символом, имеющим код Юникода nnnn
\x	Экранирует следующий символ x (где x не является алфавитно-цифровым символом). Интерпретирует x как литерал
прочие символы	Соответствует этим символам буквально

Имеется возможность описать соответствие группе или классу символов, используя одно из сокращенных обозначений символьных классов или собственный символьный класс, как показано в табл. А.4.

Таблица А.4. Символьные классы

Шаблон	Описание
.	Совпадает с любым символом (кроме символов перевода строки)
[abc]	Совпадает с любым из символов, перечисленных в квадратных скобках. Специальные символы теряют свое специальное назначение внутри символьных классов и не требуют экранирования
[^abc]	Совпадает с любым символом, кроме перечисленных в квадратных скобках
[a-z]	Совпадает с любым символом в диапазоне от a до z
[0-9A-Za-z]	Совпадает с любым алфавитным или цифровым символом ASCII
\w	Совпадает с любым символом слова (алфавитный символ, цифра, подчеркивание). Эквивалентно классу [0-9A-Za-z_]
\W	Совпадает с любым символом, кроме символа слова (то есть кроме перечисленных в предыдущем шаблоне). Эквивалентно классу [^0-9A-Za-z_]
\d	Совпадает с любой цифрой (от 0 до 9). Эквивалентно классу [0-9]
\D	Совпадает с любым нецифровым символом (то есть кроме перечисленных в предыдущем шаблоне). Эквивалентно классу [^0-9]
\s	Совпадает с любым пробельным символом (пробел, табуляция, перевод формата, перевод строки и т. д.). Эквивалентно классу [\f\n\r\t\v\u00A0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\ u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]
\S	Совпадает с любым непробельным символом (то есть кроме перечисленных в предыдущем шаблоне). Эквивалентно классу [^ \f\n\r\t\v\u00A0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\ u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]

Ограничить позицию совпадения шаблона можно с помощью конструкций, перечисленных в табл. А.5. Выражение `E` в этой таблице представляет произвольное регулярное выражение.

Таблица А.5. Привязка совпадения к определенной позиции

Шаблон	Описание
`^E`	Совпадение с выражением `E` находится в начале строки. Например, выражение `^foo` найдет совпадение `foo` в строке `food`, но не найдет в строке `junk food`
`E$`	Совпадение с выражением `E` находится в конце строки. Например, выражение `bar$` найдет совпадение `bar` в строке `rebar`, но не найдет в строке `embargo`
`\b`	Совпадает с границей слова (то есть с позицией между символом слова и пробелом, символом перевода строки, знаком пунктуации, а также началом или концом строки). Исключение составляет случай использования этого метасимвола в символьных классах (внутри квадратных скобок), где он соответствует символу забоя (backspace). Например, выражение `\\bion\\b` найдет совпадение `ion` в строке `positive ion`, но не найдет в строке `additional info`
`\B`	Совпадает с любой позицией, не являющейся границей слова (в отличие от предыдущего шаблона). Например, выражение `\\Bion\\B` найдет совпадение `ion` в строке `additional info`, но не найдет в строке `positive ion`

При необходимости можно указать, что совпадение с некоторым символом в шаблоне не является обязательным или может/должно повторяться, с помощью квантификаторов, перечисленных в табл. А.6. Без любого из этих метасимволов каждый литерал символа будет соответствовать точно один раз. Выражение `E` в этой таблице представляет произвольное регулярное выражение.

Таблица А.6. Повторение совпадений

Шаблон	Описание
`E?`	Выражение `E` совпадает ноль или один раз. Например, выражение `ba?r` найдет совпадение с `br` в строке `broom` и с `bar` в строке `embargo` , но не найдет совпадение с `baaaar` в строке `baaaargain`
`E*`	Выражение `E` совпадает ноль или более раз. Например, выражение `ba*r` найдет совпадение с `br` в строке `broom` и с `bar` в строке `embargo` , и с `baaaar` в строке `baaaargain`
`E+`	Выражение `E` совпадает один или более раз. Например, выражение `ba+r` найдет совпадение с `bar` в строке `embargo` и с `baaaar` в строке `baaaargain`, но не найдет совпадение с `br` в строке `broom`
`E{n}`	Выражение `E` совпадает точно `n` раз. Например, выражение `t{2}` найдет совпадение с `tt` в строке `committee`, но не найдет совпадение с `t` в строке `title`

Таблица А.6 (окончание)

Шаблон	Описание
E{*n*, *m*}	Выражение E совпадает от n до m раз. Например, выражение t{1,2} найдет совпадение с tt в строке committee и с t в строке title
E{*n*,}	Выражение E совпадает n и более раз. Например, выражение t{2,} найдет совпадение с tt в строке committee, но не найдет совпадение с t в строке title

Поддержка альтернативных вариантов, возможность группировки и сохранения фрагментов совпадений позволяют создавать более сложные регулярные выражения, как показано в табл. А.7. Как и прежде, выражения E и F в этой таблице представляют произвольные регулярные выражения.

Таблица А.7. Альтернативные варианты и группировка

Шаблон	Описание
E|*F*	Ищет совпадение с выражением E или F. Может включать дополнительные альтернативы. Например, выражение ise|ize найдет совпадение и в строке localise, и в строке localize, но не найдет в строке localisation
(*E*)	Сохранит совпадение с выражением E
(*E*|*F*)	Сохранит совпадение с выражением E или с выражением F
(?:*E*)	Совпадет с выражением E, но не сохранит совпадение
E(?=*F*)	Совпадет с выражением E, если за ним следует выражение F. Например, выражение one(?= two) найдет совпадение с one в строке one two, но не найдет в строке one of
E(?!*F*)	Совпадет с выражением E, если за ним не следует выражение F. Например, выражение one(?! two) найдет совпадение с one в строке one of, но не найдет в строке one two

На сохраненные фрагменты можно ссылаться в последующих частях выражения, как описывается в табл. А.8. С помощью этой конструкции можно гарантировать присутствие одиночной или двойной кавычки в конце строки.

Таблица А.8. Обратные ссылки

Шаблон	Описание
от \1 до \9	Совпадает с прежде сохраненным фрагментом в группе. Отсчет групп начинается с 1. Например, выражение (["']) (.*)\1 найдет совпадение с "real" в строке "real" world, но не найдет в строке 'real" world

A.3. Функции объекта RegExp

Объект RegExp имеет несколько функций, с помощью которых можно применять регулярное выражение, содержащееся внутри объекта.

- compile(pattern, modifiers) – компилирует или перекомпилирует регулярное выражение. Эта функция используется для изменения регулярного выражения внутри объекта. Например, заменить слово *man* на *person*, а затем слово *woman* на *person* можно следующим образом:

```
var re = /\bman\b/g;
text = text.replace(re, 'person');
re.compile(/\bwoman\b/g);
text = text.replace(re, 'person');
```

- exec(string) – применяет регулярное выражение к указанной строке и возвращает первое найденное совпадение или null, если совпадение не было найдено. Каждое совпадение представлено массивом, в элементе [0] которого хранится совпадение со всем регулярным выражением, а в остальных элементах – фрагменты совпадений, сохраненные группами в круглых скобках. Например, извлечь название протокола и имя хоста из допустимого адреса URL можно следующим образом:

```
var re = /^(http|https):\/\/([^\/]+).*/;
var match = re.exec(text);
if (match) {
    alert('protocol: ' + match[1] + ', host: ' + match[2]);
}
```

  Обратите внимание, что эту функцию можно многократно применить к одной и той же строке (добавив модификатор g), в результате чего она будет постепенно продвигаться к последнему совпадению, позволяя вам обрабатывать промежуточные совпадения.

- test(string) – применяет регулярное выражение к указанной строке и возвращает true, если в строке найдено совпадение, или false – если нет. Например, определить, начинается ли строка с последовательности символов http: или https:, можно следующим образом:

```
var re = /^(http|https):/;
if (re.test(text)) {
    ...
}
```

А.4. Функции объекта String

Некоторые функции объекта `String` также позволяют использовать регулярные выражения.

- `match(re)` – применяет регулярное выражение к текущей строке и возвращает массив совпадений или `null`, если ни одного совпадения не найдено. Возвращаемый массив будет содержать все совпадения со всем регулярным выражением, если объект регулярного выражения `re` был создан с модификатором `g`. В противном случае массив будет содержать только первое совпадение в нулевом элементе, а в остальных – фрагменты, сохраненные группами в круглых скобках. Например, извлечь название протокола и имя хоста из допустимого адреса URL можно следующим способом:

```
var re = /^(http|https):\/\/([^\/]+).*/;
var match = text.match(re);
if (match) {
    alert('protocol: ' + match[1] + ', host: ' + match[2]);
}
```

  Обратите внимание, что эта функция напоминает функцию `exec` объекта `RegExp`, но вызывается относительно строки.

- `replace(re, replacement)` – применяет регулярное выражение (`re`) и замещает найденные совпадения значением `replacement`, возвращая измененную строку. Оригинальная строка при этом не изменяется.

  Аргумент `replacement` может содержать обратную ссылку в форме `$n`, где *n* – номер соответствующей пары круглых скобок в выражении, ограничивающих (сохраняющую) группу. Например, поменять местами имя и фамилию можно с помощью обратных ссылок, как показано ниже:

```
var re = /^(\w+)\s+(\w+)$/;
text = text.replace(re, '$2, $1');
```

  Аргумент `replacement` также может быть ссылкой на функцию, принимающую текст каждого совпадения, сопровождаемый сохраненными фрагментами, совпавшими с группами в круглых скобках, и возвращающую замещающий текст. Например, преобразовать символы нижнего регистра в их эквиваленты в верхнем регистре можно следующим образом:

```
var re = /[a-z]/g;
text = text.replace(re, function(lower) {
    return lower.toUpperCase();
});
```

- `search(re)` – применяет регулярное выражение и возвращает позицию первого совпадения в строке или `-1`, если совпадение не найдено. Например, найти позицию первого числа в строке можно следующим образом:

```
var index = text.search(/\d/);
```

- `split(re)` – разбивает строку в массив, используя регулярное выражение в качестве разделителя. В качестве разделителя можно также использовать простую строку. Например, разбить строку по запятым (`,`) или символам табуляции можно, как показано ниже:

```
var fields = text.split(/[,\t]/);
```

A.5. Приемы применения

Решение некоторых типичных задач программирования становится особенно простым при использовании регулярных выражений. В этом разделе вашему вниманию будут представлены некоторые примеры таких решений.

A.5.1. Проверка данных

С помощью регулярного выражения можно проверить значение поля ввода на соответствие некоторому шаблону и при необходимости вывести сообщение об ошибке. Например, убедиться, что поле ввода содержит действительный номер социального страхования США, можно следующим образом:

```
var ssnRE = new RegExp('^\\d{3}-\\d{2}-\\d{4}$');
if (!ssnRE.test($('#ssn').val())) {
    alert('Invalid SSN');
}
```

Обратите внимание, что в этом примере используется регулярное выражение в виде строки, поэтому в нем необходимо экранировать символ обратного слэша (`\`) в метасимволе `\d`.

А.5.2. Извлечение информации

Еще одной областью применения регулярных выражений является извлечение информации из текста с использованием сохраняющих групп, заключенных в круглые скобки внутри регулярного выражения. Например, разбить адрес URL на составляющие его компоненты – протокол (перед ://), имя хоста, необязательный номер порта (с предшествующим двоеточием), путь и имя файла (после последнего /) – можно следующим образом:

```
var urlRE = /^(.*):\/\/([^/:]*)(:\d+)?\/(.*)\/(.*)$/;
var matches = url.match(urlRE);
alert('Protocol: ' + matches[1] + ', server: ' + matches[2] + ', port: ' +
    matches[3] + ', path: ' + matches[4] + ', file: ' + matches[5]);
```

А.5.3. Обработка нескольких совпадений

Обойти и обработать по отдельности все совпадения с шаблоном, имеющиеся в строке, можно с помощью функции `exec`, которая запоминает позицию последнего найденного совпадения. Например, ниже показано, как можно обработать строку, содержащую множество чисел, за каждым из которых следует символ, определяющий смысловое значение числа (`y` – год, `o` – месяц, `w` – неделя и т. д.):

```
var re = /([+-]?\d+)\s*([yowdhms])/gi;
var match;
while (match = re.exec(text)) {
    switch (match[2]) {
        case 'y': case 'Y':
            ... // обработать match[1] как год
            break;
        case 'o': case 'O':
            ... // обработать match[1] как месяц
            break;
        case 'w': case 'W':
            ... // обработать match[1] как номер недели в году
            break;
        case 'd': case 'D':
            ... // обработать match[1] как номер дня в месяце
            break;
        case 'h': case 'H':
            ... // обработать match[1] как часы
            break;
        case 'm': case 'M':
            ... // обработать match[1] как минуты
            break;
        case 's': case 'S':
            ... // обработать match[1] как секунды
            break;
    }
}
```

А.6. В заключение

Регулярные выражения – один из важнейших инструментов, с точки зрения увеличения эффективности программного кода на JavaScript. С их помощью можно проверить формат строкового значения или извлечь необходимую информацию из него. Чтобы освоить синтаксис регулярных выражений, требуется время, но эти затраты окупятся сторицей.

Глоссарий

$

Переменная **JavaScript**, используемая в библиотеке **jQuery** как синоним объекта `jQuery`.

ActiveX

Фреймворк компании Microsoft для разработки программных компонентов многократного пользования, независимых от конкретного языка программирования.

Ajax

Asynchronous JavaScript and XML (асинхронный JavaScript и XML). Технология разработки веб-приложений, применяемая для реализации отправки данных на сервер и получения их от сервера асинхронно (в фоновом режиме) без влияния на внешний вид и поведение существующей веб-страницы.

API

Application Programming Interface (прикладной программный интерфейс). Протокол, предназначенный для использования в качестве интерфейса с целью организации взаимодействий между программными компонентами.

Base62, кодировка

Схема кодирования, в которой строки символов представлены в виде последовательностей чисел в системе счисления по основанию 62. Роль цифр в этих числах играют символы 0–9, a–z и A–Z.

Behaviour

Библиотека **JavaScript**, послужившая идеей для создания библиотеки **jQuery**.

Boolean

Тип данных, имеющий два значения, `true` и `false`.

Canvas

Элемент **HTML5**, позволяющий динамически создавать и отображать двухмерные изображения.

CDN

Content Delivery Network (сеть доставки содержимого). Большая распределенная система высокопроизводительных и высокодоступных серверов, разворачиваемых для доставки содержимого конечным пользователям.

Chrome

Свободно распространяемый веб-браузер, разрабатываемый компанией Google.

Cookie

Обычно небольшой фрагмент данных, отправляемый веб-сайтом и хранящийся в браузере на стороне клиента. Автоматически отправляется обратно этому же веб-серверу при каждом последующем посещении сайта.

CSS

Cascading Style Sheets (каскадные таблицы стилей). Определения стилей, хранящиеся отдельно от разметки **HTML**.

CSV

Comma-separated values (значения, разделенные запятыми). Формат файлов для хранения табличных данных (чисел и текста) в простой текстовой форме, где поля отделяются друг от друга запятыми.

Deferred

Вспомогательный объект из библиотеки **jQuery**, позволяющий регистрировать множество функций обратного вызова в виде очередей, вызывать эти функции целыми очередями и передавать им признак успеха или неудачи, полученный в результате выполнения любой синхронной или асинхронной функции.

DOM

Document Object Model (объектная модель документа). Модель документа HTML, упрощающая выполнение операций с ним в **JavaScript**.

Ease-in

Ускорение. Разновидность **функции управления переходом**.

Ease-out

Замедление до остановки. Разновидность **функции управления переходом**.

Ease-in-out

Комбинация ускорения и замедления, когда скорость изменения сначала маленькая, затем увеличивается, после чего замедляется до полной остановки.

Firebug

Инструмент веб-разработчика, упрощающий отладку, правку и мониторинг **CSS**, **HTML**, **DOM**, **XHR** и **JavaScript**. Дополнение к **Firefox**.

Firefox

Веб-браузер, распространяемый с открытыми исходными текстами, разрабатываемый организацией Mozilla Foundation.

GZip

Прикладное программное обеспечение, выполняющее сжатие/разжатие.

HTML

Hypertext Markup Language (язык разметки гипертекста). Используется для определения содержимого веб-страниц.

IE

См. **Internet Explorer**.

Internet Explorer

Веб-браузер с графическим интерфейсом, разрабатываемый корпорацией Microsoft и входящий в состав линейки операционных систем Microsoft Windows.

Java

Многоцелевой, объектно-ориентированный язык программирования, первоначально созданный компанией Sun Microsystems.

JavaScript

Язык сценариев, обычно реализованный как составная часть веб-браузера с целью дать возможность создавать веб-страницы с динамическим пользовательским интерфейсом.

jQuery

Небольшая и быстрая библиотека **JavaScript**, упрощающая обход элементов документа **HTML**, обработку событий, воспроизведение анимационных эффектов и выполнение операций с применением технологии Ajax.

jQuery UI

Отдельный проект, основанный на библиотеке **jQuery**. Основной его целью является создание универсальных **виджетов пользовательского интерфейса**, отличающихся единообразием внешнего вида и поведения.

JSON

JavaScript Object Notation (форма записи объектов JavaScript). Легковесный формат обмена данными, основанный на подмножестве языка программирования **JavaScript**. См. http://www.json.org/[1].

MooTools

Библиотека **JavaScript**, похожая на **jQuery**. См. http://mootools.net/.

.Net

Программный фреймворк, созданный корпорацией Microsoft, включающий огромную библиотеку и обеспечивающий возможность взаимодействия программ, написанных на разных языках.

Prototype

Библиотека **JavaScript**, похожая на библиотеку jQuery. См. http://www.prototypejs.org/.

QUnit

Мощная и простая в использовании инфраструктура модульного тестирования программного кода на **JavaScript**, используемая командой проекта **jQuery**. См. http://qunitjs.com/.

RGB

Red/Green/Blue (красный/зеленый/синий). Кодировка значения цвета, определяющая величины трех составляющих цвета.

---

[1] http://ru.wikipedia.org/wiki/JSON – *Прим. перев.*

Rhino

Интерпретатор **JavaScript**, распространяемый с открытыми исходными текстами. См. www.mozilla.org/rhino.

Safari

Веб-браузер, разрабатываемый компанией Apple.

script.aculo.us

Библиотека **JavaScript**, похожая на библиотеку **jQuery**. См. http://script.aculo.us/.

Sizzle

Механизм поддержки **селекторов**, встроенный в библиотеку **jQuery**. См. http://sizzlejs.com/.

SSN

Social Security Number (номер социального страхования). Девятизначный номер социального страхования граждан США, постоянных жителей и временных работников.

ThemeRoller

Инструмент для создания собственных **тем jQuery UI** с цель интеграции их в свои проекты. См. http://jqueryui.com/themeroller/.

this

Зарезервированная переменная **JavaScript**, определяющая текущий контекст функции.

UI

User interface (пользовательский интерфейс).

URL

Uniform resource locator (унифицированный указатель ресурса; первоначально назывался universal resource locator – универсальный указатель ресурса). Строка символов, определяющая адрес ресурса в Интернете.

XML

Extensible Markup Language (расширяемый язык разметки). Язык отображения документов с иерархической структурой в простой текст.

XHR

См. **XMLHttpRequest**.

XMLHttpRequest

Встроенный объект **JavaScript**, поддерживающий возможность использования технологии **Ajax**.

Zip

Формат файлов, используемый для сжатия и архивирования данных.

Виджет (widget)

Синоним **расширения UI**. Обычно этим термином обозначаются модули **jQuery UI**.

Замыкание (closure)

Выражение (обычно функция), имеющее собственные переменные вместе с окружением, связанным с этими переменными (то есть «замкнутое» выражение). См. http://jibbering.com/faq/notes/closures/[1].

Инкапсуляция

Механизм языка программирования для ограничения доступа к некоторым компонентам объектов и языковая конструкция, упрощающая связывание данных с методами (или другими функциями). См. http://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)[1].

Локализация

Адаптация приложения под разные языки и национальные особенности.

Метод

Дополнительная функция, вызываемая расширением при передаче имени метода главной функции расширения; например: `$('#tabs').tabs('disable')`.

---

[1] http://ru.wikipedia.org/wiki/Замыкание_(программирование) – *Прим. перев.*

[1] http://ru.wikipedia.org/wiki/Инкапсуляция_(программирование) – *Прим. перев.*

Минификация кода

Уменьшение объема программного кода за счет удаления лишнего текста: комментариев и пробелов.

Модульный тест (unit test)

Группа тестов, подтверждающих работоспособность модуля/**расширения** как самостоятельной единицы (unit).

Область видимости

Контекст внутри программы, где без ограничений можно использовать любые имена переменных и другие идентификаторы.

Одиночка (singleton)

Шаблон проектирования, когда в глобальной области видимости может существовать только один экземпляр объекта.

Проверка (assertion)

Инструкция, определяющая ожидаемый результат в **модульных тестах**.

Проверка (validation)

Проверка корректности значений элементов перед отправкой на сервер.

Пространство имен

Абстрактный контейнер или окружение, созданный для хранения логически сгруппированных уникальных идентификаторов или символов (имен).

Расширение

Упакованный сценарий, встраиваемый в библиотеку **jQuery** посредством одной из точек интеграции так, что его возможности могут использоваться подобно встроенным возможностям библиотеки.

Расширение коллекций

Расширение для jQuery, оперирующее коллекцией элементов, выбранных с помощью селектора или в результате обхода дерева DOM. Большинство сторонних расширений для jQuery относится к этому типу.

Расширение-функция

Расширение для jQuery, которое не оперирует множеством выбранных элементов **DOM**, а является вспомогательной функцией.

Регулярное выражение

Краткое и гибкое средство «сопоставления» строк текста с шаблонами, такими как определенные символы, слова или комбинации символов. В английском языке для обозначения регулярных выражений часто используются сокращения *regex* и *regexp*.

Рефакторинг

Прием реструктуризации существующего программного кода с целью изменения его внутренней организации без влияния на его внешнее поведение.

Селектор

Шаблон, используемый для поиска элементов DOM и их извлечения с целью дальнейшей обработки.

Селектор псевдокласса

Селектор, выбирающий элементы, опираясь на другие признаки элементов, отличные от их имен, атрибутов или содержимого.

Танцы

Нечто, что сложно описать словами. Это надо исполнять.

Тема

Комплекс стилей оформления внешнего вида **виджетов jQuery UI**.

Цепочки вызовов

Парадигма программирования в **jQuery**, согласно которой функции должны возвращать текущее множество выбранных элементов, чтобы к нему можно было применить другие функции.

Фильтр

Еще одно название **селекторов jQuery**.

Функция обратного вызова (callback)

Ссылка на фрагмент выполняемого кода, которая передается как аргумент другому коду, позволяет низкоуровневому программно-

му слою вызвать функцию, объявленную позднее на более высоком уровне. В **JavaScript** функции обратного вызова часто используются для обработки асинхронных событий.

Функция управления переходом (easing)

Ускоряет или замедляет изменение объекта. Используется механизмом анимации для управления скоростью изменения значения атрибута.

Экранирование

Использование символа, обеспечивающего альтернативную интерпретацию следующих за ним символов.

Эффект

Предварительно упакованная реализация анимационного эффекта для использования с элементами веб-страницы.

Алфавитный указатель

Символы

A

B

C

D

K

L

M

N

М

О

П

P

C

T

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: 123242, Москва, а/я 20 или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. (499) 725-54-09, 725-50-27; электронный адрес **books@alians-kniga.ru**.

Кит Вуд

Расширение библиотеки jQuery

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Кисилев А. Н.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Подписано в печать 18.01.2014. Формат 60×90 1/16 .
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 32. Тираж 200 экз.
№

Веб-сайт издательства: www.дмк.рф