Готовые решения для веб-разработчиков

HTML5

РЕЦЕПТЫ ПРОГРАММИРОВАНИЯ

Кристофер Шмитт
Кайл Симпсон

O'REILLY®

ПИТЕР®

Christopher Schmitt
and Kyle Simpson

HTML5 Cookbook

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Кристофер Шмитт
Кайл Симпсон

HTML5

РЕЦЕПТЫ ПРОГРАММИРОВАНИЯ

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2012

Кристофер Шмитт, Кайл Симпсон

HTML5. Рецепты программирования

Серия «Бестселлеры O'Reilly»

Перевели с английского А. Лузган, Е. Шикарева

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Юрченко*
Ведущий редактор	*Ю. Сергиенко*
Литературный редактор	*Н. Гринчик*
Художественный редактор	*Л. Адуевская*
Корректор	*В. Листова*
Верстка	*Л. Родионова*

ББК 32.988.02-018.1 УДК 004.43

Шмитт К., Симпсон К.

Ш73 HTML5. Рецепты программирования. — СПб.: Питер, 2012. — 288 с.: ил.

ISBN 978-5-459-01265-1

Эта книга, представляющая собой сборник рецептов и готовых решений, позволит вам получить практический опыт работы с основными элементами HTML5. Издание охватывает широкий круг вопросов: от семантической разметки, веб-форм и мультимедийных элементов до технологий геолокации и JavaScript API.

Каждый рецепт, рассматриваемый в книге, включает в себя задачу, пример кода и подробное описание решения. Книга идеально подходит для веб-программистов начального и среднего уровней, которые хотят быстро освоить практические приемы применения HTML5 в веб-разработках.

ISBN 978-1449396794 англ.

ISBN 978-5-459-01265-1

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.
Налоговая льгота – общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 – литература учебная.
Подписано в печать 01.06.12. Формат 70x100/16. Усл. п. л. 23,220. Тираж 2000. Заказ № 468.
Отпечатано с готовых диапозитивов в ГППО «Псковская областная типография». 180004, Псков, ул. Ротная, 34.

Оглавление

Предисловие

Мы знаем, что вы хотите познакомиться со всеми замечательными и захватывающими новшествами HTML5, такими как веб-формы, элемент **canvas** и локальные хранилища. Но мы также знаем, насколько важен хороший старт для успешного развития. Для начала дадим определение понятию HTML5.

Что такое HTML5?

HTML5 — это спецификация (http://dev.w3.org/html5/spec/), разработанная в рамках World Wide Web Consortium (W3C) (Консорциум Всемирной паутины). На момент написания этой книги спецификация HTML5 официально считается рабочим проектом. Это значит, что в нее могут быть внесены дополнительные правки, прежде чем она станет рекомендацией. Затем рекомендация пройдет через формальный процесс утверждения, результатом которого станет определенная версия языка разметки.

Между тем, независимо от W3C, сообщество Web Hypertext Application Technology Working Group (WHATWG) занимается развитием спецификации HTML (http://whatwg.org/html). Обратите внимание, что мы не указали номер версии. Это потому, что WHATWG в последнее время решило изменить тактику и отказалось от упоминания версии. «Живой стандарт» — так в настоящее время обозначается HTML (http://blog.whatwg.org/html-is-the-new-html5). Новая модель развития говорит о том, что HTML определяется по мере разработки, а не как отдельная версия в текущий момент времени.

Поддержка функций, а не версии браузера

Что это означает для нас — дизайнеров и разработчиков? Такой подход позволяет уделять больше внимания особенностям сайта, а не полной

спецификации. Это как различие между фразами «на этом сайте есть вебсокеты и геолокация» и «это сайт на HTML5».

С другой стороны, некоторые специалисты в этой отрасли утверждают, что для валидации и поддержки эффективности сайтов дизайнерам и разработчикам *необходима* стабильная спецификация. Кроме того, наличие постоянной рабочей версии делает процесс разработки и обучения более управляемым.

В конце концов, все это просто могло дать гикам очередную пищу для споров (кто круче — Бэтмен или Человек-Паук). Будем иметь в виду существование *двух* моделей развития, а также помнить о довольно интересной политике совершенствования спецификации.

Живой пятый

Хотя сообщество WHATWG воспринимает HTML как живой документ, которому не нужен номер версии, в книге мы предпочли использовать термин *HTML5*. Почему? Единообразие важно при регулярном проектировании, понимании и реализации особенностей языка.

Все книги этой серии предназначены для того, чтобы предоставить читателю практические рецепты, которые сразу можно использовать, поэтому поговорим о поддержке HTML5. Надо отметить, что все современные браузеры в той или иной степени поддерживают новую версию языка. Но, как и его предшественники, HTML5 не гарантирует поддержку 100 % браузеров, о которой наверняка мы так и будем всегда только мечтать.

Применение JavaScript

В HTML5 появилось много интересных функций разметки, которые будут описаны в этой книге. Кроме того, новая версия поддерживает множество смежных веб-технологий, большинство из которых опираются на разнообразные JavaScript API, применяемые на веб-страницах.

Стремясь показать вам все достоинства HTML5 и предлагаемых смежных технологий, мы не станем избегать упоминания JavaScript. Порой в этой книге мы будем рассматривать довольно сложные примеры на JavaScript, обсуждать их дополнительные функции, отличающиеся от доступных в HTML5.

Если вы никогда не работали с JavaScript, то сейчас самое время восполнить этот пробел. Если вы настроены серьезно использовать HTML5, то почти наверняка захотите задействовать хотя бы часть из того, что может предложить JavaScript.

Важно отметить, что многие из рассматриваемых API в момент написания, редактирования и публикации книги еще развивались. Некоторые из них уже укомплектованы и поэтому, вероятно, будут стабильнее. Другие интерфейсы все еще находятся в подвешенном состоянии — имейте это в виду, когда будете решать, как использовать на своих страницах технологии HTML5.

Что вы найдете в книге

Мы не можем предоставить вам полный список функций HTML5 и браузеров, обеспечивающих их работу. Но мы расскажем о поддержке каждого элемента, рассмотренного в книге, и приведем пример его реализации (какие части спецификации браузеры поддерживают в настоящее время, вы научитесь выяснять в рецепте 2.3).

Мы также рассмотрим обходные пути решения отдельных задач, необходимые при частичной поддержке браузером. Таким образом, вы сможете определиться с функциями, полезными для вас, вашего клиента или работодателя.

Это все, что мы можем предложить, дорогой читатель. Совершенно не обязательно применять только HTML5. Не нужно добавлять встроеный контент или веб-сокеты, чтобы использовать новые структурные элементы, и наоборот, не требуется внедрять новые структурные элементы, чтобы получить HTML5-документ. Достаточно лишь определить тип документа.

Если вы не знаете, что такое определение типа документа, начните свое путешествие по HTML5 с рецепта 1.1!

Выберите функции HTML5, необходимые вам для работы, и отбросьте остальные. Или *поэкспериментируйте* с ними, чтобы увидеть, получится ли найти им практическое применение в ваших проектах.

HTML предоставляет огромное количество возможностей, и, вероятно, так будет всегда, ведь веб-технологии постоянно меняются. А мы, как хорошие веб-дизайнеры и разработчики, должны и впредь не отставать от последних изменений и событий в своей области, знакомить клиентов и работодателей с возможными преимуществами и компромиссами, экспериментировать при решении задач и постоянно совершенствовать свои навыки.

Нам кажется, что все это довольно интересно. Итак, приступим?

Аудитория

Достаточно сказать, что эта книга подойдет любому, кто заинтересован в изучении HTML5, но особенно она понравится веб-разработчикам, желающим перейти от XHTML или HTML4 к новым технологиям.

Последние главы книги в большей степени ориентированы на специалистов, которые хотят воспользоваться функциями JavaScript API, поддерживаемыми в HTML5.

Что дает эта книга

Вряд ли вам нужно краткое изложение спецификации HTML5. Скорее всего, вы хотите найти решение задачи, поставленной на работе. В каждом из рецептов книги можно получить быстрый и ненавязчивый ответ на какой-либо вопрос. Обратите внимание на подраздел «Обсуждение», присутствующий в каждом рецепте и содержащий развернутую информацию о проблеме.

Вы можете начать чтение с первой страницы и не пропустить ничего, кроме одной или двух взаимоисключающих частей рецепта. Однако главным нашим желанием при написании книги было стремление помочь вам справиться с конкретной проблемой, в данном случае связанной с HTML5. Просто откройте книгу, и вы найдете необходимый рецепт с решением, подходящим для практического использования.

И еще кое-что: мы предполагаем, что HTML5 всегда будет поддерживать геолокацию. Хотя рабочий проект Geolocation API W3C существует отдельно от спецификации HTML5, это не останавливает людей, ведущих блоги, пишущих учебники и даже книги о применении геолокации в HTML5. И поскольку этот API сейчас становится все более популярным, мы решили добавить в книгу соответствующий раздел (подробнее в главе 8).

Условные обозначения

В издании используются следующие условные обозначения.

Шрифт для элементов интерфейса

Применяется для отображения URL, названий папок и выводимой на экран информации.

Шрифт для команд

Используется для выделения фрагментов компьютерного кода. Кроме того, внутри разделов им обозначены программные элементы, такие как переменные или названия функций, типы, операторы и ключевые слова.

Курсивный шрифт для команд

Обозначает текст, который должен быть заменен пользовательским значением или значением, определенным контекстом.

Моноширинный шрифт

Применяется для примеров исходного кода и текста файлов.

Полужирный моноширинный шрифт

Применяется для команд или текста, который пользователь должен набирать вручную.

Такой значок обозначает совет, предложение или замечание общего характера.

Этот значок обозначает предупреждение или предостережение.

Использование примеров кода

Задача этой книги — помочь вам справиться с конкретной задачей. Вы можете добавлять приведенный код в свои программы и документацию. Для этого вам не требуется наше разрешение. Чтобы написать программу, использующую несколько фрагментов кода из этой книги, вам не нужно обращаться к нам. Разрешение не требуется, если вы, отвечая на вопрос,

процитировали книгу или фрагмент кода. Если же вы хотите включить значительное количество примеров кода из книги в документацию по своему продукту, вам понадобится наше разрешение.

Если вам кажется, что вы заимствуете слишком большой объем программного кода, не стесняйтесь обращаться в издательство.

От издательства

У книги есть веб-страница, на которой содержатся исправления, примеры и другие дополнительные сведения. Вы можете получить доступ к ней по адресу http://shop.oreilly.com/product/0636920016038.do.

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Благодарности

Мы не замечаем, но время — это невидимое измерение, которое окружает нас. Время — явление не только очень интересное, но и прекрасное, оно позволяет определять даты окончания и сроки, в которые должна быть издана книга.

Авторам устанавливается крайний срок, к которому они должны написать книгу. Множество талантливых людей — литературные и научные редакторы, верстальщики, художники, дизайнеры — могут вносить в рукопись изменения. Каждый из них выставляет для автора свои сроки — сроки на редактирование, технические правки, верстку, художественное оформление и т. д.

Сроки за сроками — это и есть время.

Но есть и другой способ обозначить время.

Время вдали от родных и близких.

После написания нескольких книг я понял, что отказываться от радостей бытия, будь то праздники или общение с семьей, — это поистине подвиг, и я не пожелал бы этого никому.

Но у нас есть друзья, не так ли?

Вместе с моим соавтором Кайлом Симпсоном я поздравляю и благодарю за вклад в «HTML5: Рецепты программирования»: Эмили Льюис, Кимберли Блессинг, Кристину Хаггинс Рэми, Анитру Павка, Марка Грабански и Кристофера Дойтча.

Спасибо нашим техническим редакторам Шелли Пауэрс (Shelley Powers), Бену Хенику (Ben Henick), Дасти Джевитту (Dusty Jewett), Молли Хольцшлаг (Molly Holzschlag) и Хелен Оливер (Helen Oliver), которые старательно держали нас в тонусе.

Нашему верстальщику Рэйчалу Хэду (Rachel Head) — чудотворцу, который заставил наши семьи гордиться нами и думать, что все мы куда более талантливые писатели, чем, вероятно, есть на самом деле.

Огромное спасибо редакторам O'Reilly Симону Сен-Лорану (Simon St. Laurent) и Кортни Нэш (Courtney Nash), которые во время написания этой книги помогли мне сохранить рассудок.

Мне чрезвычайно сложно представить, каким огромным опытом обладают веб-разработчики, привлеченные к этому проекту и проявившие все свои писательские и редакторские таланты.

Я не уверен, что эти одаренные люди знали, на что они подписываются, когда я впервые говорил с ними о книге «HTML5: Рецепты программирования», но я счастлив, что нам удалось создать ее вместе. И я благодарен им за это.

И наконец, я благодарен своим родным и близким, которые поддерживали меня в моей одержимости веб-дизайном и разработкой. Давайте и дальше делиться хорошими моментами, проведенными вместе.

Кристофер Шмитт
http://christopherschmitt.com
http://twitter.com/@teleject

Я хотел бы поблагодарить свою прекрасную и терпеливую жену Кристин, позволившую мне принять участие в этом проекте. Я хочу сказать спасибо своим родителям за помощь и поддержку во всех моих начинаниях. И, в конце концов, моему новорожденному сыну Итану: я надеюсь, когда-нибудь тебе выпадет счастье писать на HTML12, и ты будешь смотреть на эту старую книгу об HTML5 с ностальгией. Просто помни, что мне приходилось учиться и в снег, и в дождь и не только...

Кайл Симпсон
http://blog.getify.com
http://twitter.com/@getify

Спасибо Кристоферу Шмитту за то, что дал возможность внести вклад в эту книгу. Я также хотела бы поблагодарить Рея Банго (Rey Bango), который рекомендовал меня Кристоферу и дал шанс написать о том, что я люблю. Я должна поблагодарить Кристиана Хайльмана (Christian Heillman), который познакомил меня с Реем и всегда поддерживал мое стремление поделиться опытом.

Спасибо моему любимому Джейсону, который помогает мне во всех делах и напоминает о том, что все новое не так страшно, как кажется на первый взгляд. Благодарю тебя за то, что читаешь все, что я пишу, и убеждаешь меня, что звучит это не так уж и глупо. Ты делаешь мою жизнь полнее и лучше, чем я когда-либо мечтала.

Спасибо моим неофициальным редакторам Брайану Арнольду (Brian Arnold) и Яну Питтсу (Ian Pitts) за то, что на протяжении месяцев (или лет) они находили время почитать мою работу и предложить свою критику. Ребята, вы не только блестящие разработчики, но и прекрасные друзья.

Спасибо моей сестре Эрин Шутес (Erin Shutes), напоминавшей мне о том, кто я и что важно в этом мире. Я благодарна своей сестренке за то, что она 20 лет поддерживает меня во все лучшие и худшие времена.

Я признательна читателям моего блога, моим друзьям из социальных сетей и всем тем, кого я встречала на конференциях, за то, что они заинтересованы в обмене опытом. Они каждый день напоминают мне, почему я люблю свою работу.

Эмили Льюис
http://emilylewisdesign.com
http://twitter.com/@emilylewis

Я хотела бы поблагодарить моего друга Кристофера Шмитта за приглашение поучаствовать в написании этой книги, моих студентов из Bryn Mawr College и коллег из Comcast Media Interactive, с которыми я постоянно консультировалась, мою мать и двоюродную бабушку за их содействие, а также мою кошку Панки, которая тоже меня поддерживала.

Кимберли Блессинг
http://www.obiwankimberly.com
http://twitter.com/@obiwankimberly

Спасибо Молли Хольцшлаг и Камерон Баррет (Cameron Barrett), которые посоветовали мне поделиться своими знаниями.

Конечно же, спасибо Кристоферу Шмитту, убедившему меня внести свой вклад в написание этой книги.

Спасибо Мэтту Мэю (Matt May) за его рекомендации, которые я приняла во внимание при создании книги.

Я также хотела бы поблагодарить всех замечательных людей, с которыми я на протяжении многих лет встречалась на конференциях и мероприятиях по Сети и сетевым технологиям. Можно довольно долго перечислять их имена. Все вы заставляете меня узнавать что-то новое и радоваться.

Спасибо, маме и папе, за... пожалуй, все.

Кроме того, я хотела бы поблагодарить наших читателей — вас — за то, что вы совершенствуете свое искусство и оттачиваете свои навыки. Ваше любопытство, знания и сопереживание помогают Интернету развиваться.

Анитра Павка
http://anitrapavka.com
http://twitter.com/@apavka

Я выражаю сердечную благодарность Кристоферу Шмитту за предоставленную мне возможность расширить свои знания о веб-дизайне. Вы вывели мои навыки письма и программирования далеко за пределы моих скромных амбиций.

Мой дорогой муж Поль Реми (Paul Ramey), спасибо тебе за поддержку и терпение, которое ты проявил, пока я занималась исследованиями, программированием и писала. Я благодарна тебе за то, что ты отвлекал меня, когда я работала слишком долго!

Кристина Хаггинс Рэми
http://www.christinaramey.me
http://twitter.com/@fidlet

Спасибо Кристоферу Дойтчу за идеи по поводу главы, посвященной геолокации, и их воплощение.

Марк Грабански
http://marcgrabanski.com
http://twitter.com/@1Marc

1 Основы синтаксиса и семантики

Эмили Льюис

1.0. Введение

Эта глава предназначена для того, чтобы познакомить вас с основами HTML5. Она содержит рецепты на базовые темы: начиная с объявления **DOCTYPE** и набора символов и заканчивая проблемами, связанными с двусмысленностью новой семантики HTML5. Все это послужит фундаментом для остальных разделов книги.

1.1. Объявление DOCTYPE

Проблема

Необходимо создать HTML5-страницу.

Решение

В самом начале HTML5-страницы укажите **DOCTYPE**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML5 для работы &amp;amp; отдыха</title>
```

продолжение ⇩

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

Обратите внимание, что DOCTYPE нечувствителен к регистру. Можете смело использовать CaMeL cAsE (ВеРбЛюЖиЙ рЕгИсТр).

Обсуждение

Директива **DOCTYPE** отвечает за определение типа документа, то есть сообщает браузерам и валидаторам, какая версия HTML использовалась при написании страницы. Ранее указывался номер версии; так, к примеру, объявляется **DOCTYPE** для XHTML 1.0 Strict:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

В HTML5 версия в **DOCTYPE** не указывается, что с точки зрения синтаксиса обеспечивает для языка обратную совместимость и, надеюсь, упрощает переход на HTML5.

Допустим, у вас есть сайт, соответствующий стандарту HTML 4.0, но вы хотите перевести его на HTML5. Для этого достаточно изменить **DOCTYPE**.

Все браузеры поддерживают сокращенное объявление **DOCTYPE** и отображают страницу в строгом соответствии стандартам.

Существуют некоторые различия между элементами HTML4 и HTML5, поэтому необходимо обратить внимание на удаленные или устаревшие элементы. Например, элемент center не пройдет валидацию в HTML5.

Дополнительная информация

Обсуждение различий в объявлении типа документа между HTML4 и HTML5 для рабочего проекта W3C можно найти по адресу http://www.w3.org/TR/html5-diff/#doctype.

1.2. Определение кодировки

Проблема

Необходимо задать кодировку, используемую на вашей странице.

Решение

В своем документе поместите внутри элемента **head** элемент **meta** и укажите в нем такой набор символов:

```
<meta charset="UTF-8" />
```

Обсуждение

Кодировка сообщает браузерам и валидаторам, какой набор символов использовать при отображении веб-страницы. Если в HTML-документе кодировка не объявлена, то браузер сначала попытается определить ее через заголовок ответа вашего HTTP-сервера (в частности, заголовок **Content-Type**).

Кодировка, заданная в заголовке ответа, предпочтительнее, нежели указанная в документе, поэтому заголовки — основной способ предоставления такой информации. Однако существует возможность контролировать, какие заголовки будет отправлять ваш сервер, — для этого необходимо объявить кодировку в HTML-документе. Это оптимальный вариант.

Если кодировка не определена ни в документе, ни в ответе заголовка, то браузер попробует выбрать ее сам, что *может* не соответствовать потребностям вашего сайта. В этом случае есть риск не только вызвать проблемы с отображением страниц, но и создать угрозу безопасности.

Несколько лет назад в Google была обнаружена уязвимость, названная межсайтовым скриптингом: http://shiflett.org/blog/2005/dec/googles-xss-vulnerability. Это показало всю важность определения кодировки.

В предыдущих версиях HTML кодировка объявлялась с помощью дополнительных атрибутов и значений:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Но, как и в случае с **DOCTYPE**, в HTML5 задается только необходимая браузерам информация. Это помогает обеспечить обратную совместимость и упрощает труд разработчика.

Специальные символы. Unicode (UTF-8) представляет собой универсальную кодировку, удовлетворяющую потребностям большинства веб-разработчиков. Правда, иногда необходимо добавлять не входящие в нее символы.

Отличный ресурс со списком символьных сущностей находится по адресу http://www.digitalmediaminute.com/reference/entity/. Вы найдете числовые, именованные и Unicode-ссылки на наиболее распространенные в HTML символы.

Для того чтобы помочь браузерам правильно отображать такие символы, можно задавать их в виде именованных сущностей или числовых ссылок (NCR). К примеру, если хотите использовать символ авторского права, включите его в HTML-код как NCR:

```
&#169;
```

или в качестве именованной сущности:

```
&copy;
```

Дополнительная информация

Рассуждения Марка Пилгрима (Mark Pilgrim) о кодировке из книги «Погружение в HTML5»: http://diveintohtml5.info/semantics.html#encoding.

1.3. Определение языка

Проблема

Необходимо определить основной язык веб-страницы.

Решение

Добавьте в открывающий тег элемента **html** атрибут **lang** и присвойте ему значение:

```
<html lang="en">
```

Обсуждение

Браузеры, программы для чтения с экрана и другие пользовательские агенты применяют атрибут **lang** для определения языка интерпретации контента. В приведенном выше примере английский язык задается значением **en**.

В HTML5 (как и в любой другой из предыдущих версий) необязательно объявлять основной язык документа. Однако это хорошая практика.

Дополнительная информация

Рассуждения Марка Пилгрима об определении языка документа из книги «Погружение в доступность»: http://diveintoaccessibility.info/day_7_identifying_your_language.html.

1.4. Оптимизация <script> и <link>

Проблема

Необходимо как можно проще подключить к странице JavaScript-сценарии и вставить ссылки на внешние CSS-файлы.

Решение

Добавьте элементы **script** и **link**, но не используйте атрибут **type**:

```
<link rel="stylesheet" href="styles.css" />
<script src="scripts.js"></script>
```

Обсуждение

В HTML5 нужно указывать минимальный объем информации, необходимой пользовательским агентам. В предыдущих версиях HTML как в CSS-ссылках **link**, так и в элементе **script** для определения языка приходилось задавать атрибут **type**. Сейчас, даже если вы забыли объявить **type**, в большинстве браузеров уже предусмотрено правильное значение.

В HTML5 **type** *официально* признан необязательным атрибутом, тем не менее старые документы, содержащие его, до сих пор валидны.

Дополнительная информация

Обсуждение различий между HTML4 и HTML5 для рабочего проекта W3C относительно изменений, связанных с атрибутами, можно найти на сайте http://www.w3.org/TR/html5-diff/#changed-attributes.

1.5. Добавление в структуру документа новых элементов HTML5

Проблема

Необходимо определить структуру документа с помощью новых элементов **header**, **footer**, **nav**, **aside**, **section** и **article**.

Решение

Для того чтобы определить, какие из новых элементов необходимы на вашей странице, проанализируйте контент и структуру документа.

- **header** — используется для хранения заголовков страницы и/или разделов **section**. Может содержать дополнительную информацию, например логотипы и навигацию.
- **footer** — содержит информацию о странице и/или разделе **section**, например информацию об авторе, ссылки на использованные источники, а также сведения об авторских правах.
- **nav** — включает *основные* навигационные ссылки на страницы и зачастую содержится в **header**.
- **aside** — хранит информацию, *связанную* с окружающим контентом, но существующую *независимо*, например на боковой панели.
- **section** — основной из новых структурных элементов, контент которых может быть сгруппирован тематически или связан между собой.
- **article** — определяет автономный контент, который может использоваться независимо от страницы в целом, подобно записи в блоге.

Простая структура блога с заголовком, навигацией, боковой панелью, постами и подвалом может быть задана в HTML5 следующим образом:

```
<header>
    <h1><abbr title="Язык гипертекстовой разметки">HTML</abbr>5 для
        работы &amp;amp; отдыха</h1>
    <nav>
```

```
        <ul>
            <li><a href="/Archive/">Архив</a></li>
            <li><a href="/About/">О проекте</a></li>
        </ul>
    </nav>
</header>
<article>
    <h2><code>nav</code> нужен не для <em>всех</em> ссылок</h2>
    <p> Тот факт, что элемент <code>nav</code> зачастую содержит
        ссылки, не означает, что <em>все</em> ссылки на сайте
        необходимо помещать в <code>nav</code>.</p>
</article>
<article>
    <h2>У вас есть <code>DOCTYPE</code>. Что теперь?</h2>
    <p> HTML5 — это не все или ничего. Вы можете выбрать то, что
        будет лучше для вас. Достаточно просто верно указать
        <code>DOCTYPE</code>, чтобы все выяснить.</p>
</article>
<aside>
    <h2>Еще немного HTML5</h2>
    <p>Совершенствуйтесь в HTML5 с нашими партнерами:</p>
    <ul>
        <li><a href="http://lovinghtml5.com">Loving HTML5</a></li>
        <li><a href="http://semanticsally.com">Semantic Sally</a></li>
    </ul>
</aside>
<footer>
    <p>Авторские права &copy; 2011 <a href="http://html5funprofit.com">
        HTML5 для работы &amp; отдыха</a>. Все права защищены.</p>
</footer>
```

Выполнив HTML-код вместе с правильной CSS-разметкой, браузер отобразит ее так, как показано на рис. 1.1.

Рис. 1.1. Пример представления простой структуры блога с помощью новых элементов HTML5

Обсуждение

Новые структурные элементы разрабатывались в процессе реальной работы. Анализ более чем миллиарда веб-страниц (http://code.google.com/webstats/) показал, что CSS-атрибуты **class** и **id**, уже используемые разработчиками

для структурирования и описания контента страниц, могут быть заменены HTML-элементами:

- **header**;
- **footer**;
- **nav**.

Новые элементы HTML5 лишь отражают то, что уже делают разработчики.

Структурные элементы

Благодаря структурным элементам можно добавить в разметку более существенную семантику, а также определить ключевые особенности любого документа.

Посмотрим, насколько это важно пользователям программ для чтения и тем, кто переходит к различным областям страницы с помощью клавиатуры. Ранее подобная функциональность обеспечивалась благодаря ссылкам и ярлыкам (http://www.vdebolt.com/nmmug/flow.html), но HTML5 предоставляет специальные элементы. Стандарт Web Accessibility Initiative's Accessible Rich Internet Applications (WAI-ARIA) (доступности активных интернет-приложений) содержит рекомендации относительно того, как следует использовать HTML5 в ARIA с целью навигации (подробнее читайте в главе 2).

Тем не менее все это еще до конца не опробировано. На момент написания книги ни браузеры, ни вспомогательные технологии не ссылались на структурные элементы для обеспечения навигации.

Когда использовать структурные элементы

Знаете ли вы, когда следует применять эти новые элементы? Опять же, необходимо сосредоточиться на контенте и подумать о семантике каждого элемента. Например, многие веб-страницы включают область, которая считается «заголовком» и, возможно, хранит логотип, название компании и слоган. Это, безусловно, хороший повод использовать **header**. Элементы **section** или **aside** могут определяться в документе со своими заголовками и навигацией, которые также могут быть помещены в **header**.

Все это справедливо и для **footer**. На данный момент контент большинства страниц подходит для нового элемента **footer** — в него можно добавить сведения об авторе, авторском праве, а также сопутствующую информацию. Элементы **sections**, **articles** и **asides** могут содержать ту же информацию и, в свою очередь, включать в себя **footer**.

В заключение рассмотрим элемент **nav**. На сайте может быть множество различных ссылок, отдельные из которых предназначены для навигации, в то время как другие являются внешними.

Элемент nav подходит только для основной навигации по сайту, но не для ссылок на результаты поиска или ссылок на блоги.

Когда использовать элементы div

Как видно из примера с разметкой блога, новые структурные элементы могут заменить отдельные несемантические контейнеры **div**, которыми вы, возможно, привыкли пользоваться. Но элемент **div** остается актуален и в HTML5.

Если вам когда-нибудь понадобится элемент *строго определенного стиля*, используйте **div**. Необязательно применять новые структурные элементы только для связи с CSS.

Старайтесь концентрироваться на контенте и избегайте *неуместного* применения **div**, например, когда для решения задачи больше подходит другой элемент. Скажем, для абзаца с текстом используйте элемент **p**, а не **div**. Хотя и тот и другой — блочные элементы, в этом случае **p** имеет более существенную семантику.

Оформление структурных элементов

Все современные браузеры показывают содержимое новых элементов. Тем не менее некоторые обозреватели не могут их распознать и поэтому считают встроенными. Это вызывает серьезные проблемы с оформлением страницы.

К счастью, это нетрудно исправить. Достаточно сообщить браузеру, что ему следует рассматривать эти элементы как блочные:

```
header, footer, nav, article, aside, section {
    display: block;
}
```

Одного описания в CSS хватает для успешного применения стиля к структурным элементам. Или почти хватает. В версиях Internet Explorer (IE) до IE9 необходимо добавить в документ немного JavaScript, поскольку только так браузер сможет распознать эти элементы и позволит применить стилевое оформление:

```
<script>
    document.createElement('header');
    document.createElement('footer');
    document.createElement('nav');
    document.createElement('article');
    document.createElement('aside');
    document.createElement('section');
</script>
```

Если приходится работать с более ранними версиями IE, то для добавления в документ нового элемента HTML5 нужно просто указать соответствующий **document.createElement**. Более подробное обсуждение использования JavaScript с IE можно найти в главе 2.

Дополнительная информация

Статья Script Junkie «Новые семантические HTML5-теги сегодня»: http://msdn.microsoft.com/en-us/scriptjunkie/gg454786 (на английском языке).

1.6. Элемент article или section

Проблема

Неизвестно, какой из элементов лучше использовать на веб-странице — **article** или **section**.

Решение

Обратите внимание на назначение и семантику элементов **article** и **section** (вернитесь к рецепту 1.5).

Элемент article

Можно рассматривать **article** как особую форму **section**, которая предназначена для контента, самостоятельно существующего за пределами окружающей информации. Например, это сообщения, «объединенные» в блог.

Элемент **article** можно использовать и для другого контента, в том числе:

- видео и сопутствующих записей;
- новостных статей;
- комментариев к блогу.

Часто название статьи или сообщения в блоге хранится в URI. В этом случае контент страницы, с которой вы работаете, должен быть внутри элемента article.

В примере кода из рецепта 1.5 элементы **article** хранят отдельные сообщения в блоге:

```
<article>
    <h2><code>nav</code> нужен не для <em>всех</em> ссылок</h2>
    <p> Тот факт, что элемент <code>nav</code> зачастую содержит
        ссылки, не означает, что <em>все</em> ссылки на сайте
        необходимо помещать в <code>nav</code>.</p>
</article>
<article>
    <h2>У вас есть <code>DOCTYPE</code>. Что теперь?</h2>
    <p> HTML5 – это не все или ничего. Вы можете выбрать то, что
        будет лучше для вас. Достаточно верно указать
        <code>DOCTYPE</code>, чтобы все выяснить.</p>
</article>
```

Элемент section

Элемент **section** — один из основных новых структурных элементов, предназначенных для группировки связанных объектов. Тем не менее это *не* просто контейнер для элементов, как **div**. Содержимое такой группы должно быть *взаимосвязано*.

Применим это замечание к рецепту 1.5, где **section** можно добавить в качестве родительского для обоих элементов **article**:

```
<section>
    <article>
        <h2><code>nav</code> нужен не для <em>всех</em> ссылок</h2>
        <p> Тот факт, что элемент <code>nav</code> зачастую содержит
            ссылки, не означает, что <em>все</em> ссылки на сайте
            необходимо помещать в <code>nav</code>.</p>
    </article>
    <article>
        <h2>У вас есть <code>DOCTYPE</code>. Что теперь?</h2>
        <p> HTML5 – это не все или ничего. Вы можете выбрать то, что
            будет лучше для вас. Достаточно верно указать
            <code>DOCTYPE</code>, чтобы все выяснить.</p>
    </article>
</section>
```

Этот пример соответствует основным требованиям, предъявляемым к **section**: сгруппированы элементы с тематически связанным контентом.

Но подождите! Спецификация (http://www.w3.org/TR/html5/sections.html#the-section-element) уточняет:

Основной принцип заключается в том, что использование элемента section целесообразно, только если содержимое будет явно перечислено в структуре документа.

Структура документа связана с новой моделью *секционирования контента* в HTML5, где каждый новый структурный элемент создает свой

независимый контур. Такая схема реализуется заголовками (**h1-h6**), содержащимися в каждом элементе (см. рецепт 1.7).

Итак, если вы хотите использовать **section**, то содержимое элемента следует озаглавить. Изменим разметку рецепта 1.5, включив дополнительные разъяснения в заголовок **section**:

```
<section>
    <h1>Последние записи в блоге</h1>
    <article>
        <h2><code>nav</code> нужен не для <em>всех</em> ссылок</h2>
        <p> Тот факт, что элемент <code>nav</code> зачастую содержит
            ссылки, не означает, что <em>все</em> ссылки на сайте
            необходимо помещать в <code>nav</code>.</p>
    </article>
    <article>
        <h2>У вас есть <code>DOCTYPE</code>. Что теперь?</h2>
        <p> HTML5 – это не все или ничего. Вы можете выбрать то, что
            будет лучше для вас. Достаточно верно указать
            <code>DOCTYPE</code>, чтобы все выяснить.</p>
    </article>
</section>
```

Обсуждение

Различия между **section** и **article** могут привести к путанице, и с совершенствованием спецификации ситуация не проясняется. В большинстве случаев вам могут помочь такие рекомендации.

- Не используйте **section** только для задания стилей. Для этого лучше подойдет **div**.
- Не применяйте **section**, если для содержимого лучше подходит **header**, **footer**, **aside** или **article** (см. рецепт 1.5).
- Не используйте **section**, когда в содержимом нет естественного заголовка.

Дополнительная информация

HTML5 Doctor предлагает удобную схему выбора подходящего структурного HTML5-элемента: http://html5doctor.com/happy-1st-birthday-us/#flowchart.

1.7. Проверка структуры документа

Проблема

Необходимо показать структуру вашего документа.

Решение

Откройте свою страницу с помощью инструмента HTML5 Outliner: http://gsnedders.html5.org/outliner/.

Обсуждение

HTML5 содержит структурный алгоритм, вычисляющий строение веб-документа. Его работа основана на определении разделов, устанавливаемых новыми структурными элементами.

Документ формируется элементами **section** и **aside**, каждый из которых задает новый раздел в структуре документа, в то время как заголовки определяют структуру контента. Это называется *явным секционированием.*

Рассмотрим следующий пример разметки:

```
<section>
    <h1>Последние записи в блоге</h1>
    <article>
        <h2><code>nav</code> нужен не для <em>всех</em> ссылок</h2>
        <p> Тот факт, что элемент <code>nav</code> зачастую содержит
            ссылки, не означает, что <em>все</em> ссылки на сайте
            необходимо помещать в <code>nav</code>.</p>
    </article>
    <article>
        <h2>У вас есть <code>DOCTYPE</code>. Что теперь?</h2>
        <p> HTML5 — это не все или ничего. Вы можете выбрать то, что
            будет лучше для вас. Достаточно просто верно указать
            <code>DOCTYPE</code>, чтобы все выяснить.</p>
    </article>
</section>
```

Такая разметка создает следующую структуру документа:

1. Последние записи в блоге

1.1 **nav** нужен не для всех ссылок

1.2 У вас есть **DOCTYPE**. Что теперь?

Уровни заголовков

Независимо от версии, заголовки в HTML ранжируются, начиная от верхнего **h1** до нижнего **h6**.

Еще до появления HTML5 такая иерархия помогала выделить структуру документа. Тем не менее с приходом HTML5 благодаря секционированию элементов возросла важность ранжирования заголовков внутри определенной структуры.

Изменим предыдущий пример, задав **h6** в качестве основного заголовка **section** и **h1** как основной заголовок **article**:

```
<section>
    <h6>Последние записи в блоге</h6>
    <article>
        <h1><code>nav</code> нужен не для <em>всех</em> ссылок</h1>
        <p> Тот факт, что элемент <code>nav</code> зачастую содержит
            ссылки, не означает, что <em>все</em> ссылки на сайте
            необходимо помещать в <code>nav</code>.</p>
    </article>
    <article>
        <h1>У вас есть <code>DOCTYPE</code>. Что теперь?</h1>
        <p> HTML5 – это не все или ничего. Вы можете выбрать то, что
            будет лучше для вас. Достаточно верно указать
            <code>DOCTYPE</code>, чтобы все выяснить.</p>
    </article>
</section>
```

Для того чтобы узнать, когда начинается новый структурный раздел (а не заголовок), в HTML5 используются элементы **section** и **aside**, поэтому структура будет та же, что и в оригинальном примере:

1. Последние записи в блоге
1.1 **nav** нужен не для всех ссылок
1.2 У вас есть **DOCTYPE**. Что теперь?

Неявное секционирование

Структурные элементы в HTML5 *необязательны*, поэтому если секционирование элементов не используется, то ранжирование заголовков по-прежнему может влиять на структуру документа. Это называется *неявным секционированием*. Кроме того, чтобы сформировать структуру документа, можно одновременно задать явное и неявное секционирование.

Неявный раздел начинается с нового заголовка и позиционируется в структуре документа в соответствии с рангом относительно заголовка предыдущего раздела. Если отдельный заголовок имеет более низкий ранг, чем предыдущий, то он открывает новый структурный подраздел. Если заголовок имеет более высокий ранг, то он закрывает предыдущий раздел и открывает новый.

Рассмотрим пример, демонстрирующий, каким образом ранг заголовка обеспечивает неявное секционирование:

```
<section>
    <h1>У вас есть <code>DOCTYPE</code>. Что теперь?</h1>
    <h2>Оптимизация <code>link</code> и <code>script</code></h2>
    <p>HTML5 имеет более простой и гибкий синтаксис ...</p>
    <h2>Блочные ссылки</h2>
    <p>В HTML5 элементы <code>a</code> могут содержать блоки ...</p>
```

```
    <h3>Оглядываясь назад</h3>
    <p>В более ранних версиях HTML вам приходилось ...</p>
    <h2>Добавление структуры</h2>
    <p>HTML5 дает возможность добавить несколько новых элементов ...</p>
</section>
```

В этом случае мы получим следующую структуру:

1. У вас есть **DOCTYPE**. Что теперь?
1.1 Оптимизация **link** и **script**
1.2 Блочные ссылки
1.2.1 Оглядываясь назад
1.3 Добавление структуры

Почему это важно?

Зная, как выглядит структура документа, вы не только сможете решить, какой уровень заголовков следует применять, но и определитесь с использованием структурных элементов. Помните, мы говорили о том, что раздел должен иметь естественный заголовок (см. рецепт 1.6)? Проверка структуры документа покажет, не забыли ли вы включить заголовок, который, в свою очередь, даст вам понять, какой элемент нужно использовать с таким содержимым. Кроме того, структура документа, точно отражающая иерархию его контента, может помочь пользователям перемещаться по сайту. К сожалению, браузеры еще не реализуют подобное структурирование, и ни одна из вспомогательных технологий не использует его для навигации. Тем не менее проверка структуры документа должна стать частью процесса разработки, ведь это может существенно помочь при выборе элементов и уровней заголовков.

Дополнительная информация

Алгоритм структуризации и секционирование подробно рассмотрены на Mozilla Developer Network (сеть разработчиков Mozilla): https://developer.mozilla.org/en/Sections_and_Outlines_of_an_HTML5_documentisanexcellentresource.

1.8. Изменение структуры документа

Проблема

Есть основной заголовок страницы и подзаголовок, размеченные элементами **h1** и **h2** соответственно, но структура документа не должна включать подзаголовок.

Решение

Добавьте элемент **hgroup**, содержащий оба заголовка:

```
<hgroup>
    <h1>HTML5 для работы &amp;amp; отдыха</h1>
    <h2>Советы, рекомендации и ресурсы</h2>
</hgroup>
```

Обсуждение

Элемент **hgroup** скрывает все заголовки в структуре документа, кроме верхнего. В приведенном примере будет отображаться только «HTML5 для работы & отдыха», заключенное в **h1**.

Даже если мы поменяем порядок следования этих заголовков, верхним в структуре будет считаться тот, рейтинг которого выше (**h1**):

```
<hgroup>
    <h2>Советы, рекомендации и ресурсы</h2>
    <h1>HTML5 для работы &amp;amp; отдыха</h1>
</hgroup>
```

Дополнительная информация

Подробное обсуждение заголовков, **hgroup** и структурирования HTML5 от Эмили Льюис можно найти по адресу http://ablognotlimited.com/articles/the-beauty-of-semantic-markup-part-3-headings.

1.9. Выделение текста

Проблема

Необходимо придать выразительности тексту или подчеркнуть его важность.

Решение

Введите элемент **em**, предназначенный для выделения текста:

```
<p>Меня зовут <em>Джейн</em>, а не Джон.</p>
```

Обсуждение

Вы будете удивлены, узнав, насколько элемент **em** изменился по сравнению с тем, что было в HTML4. Разница в том, что HTML5 немного переопределяет **em** — он применяется для экспрессивно-эмоционального выделения, то есть обозначает текст, который необходимо выделить при произношении.

Переопределение элемента i. Еще одно изменение в HTML5, связанное с семантикой на уровне текста, — элемент **i** больше не считается презентационным. Теперь вы можете использовать его для дополнительного выделения слов, например технических терминов, мыслей, идиоматических фраз или иного текста, обычно набираемого курсивом:

```
<p>Салли подумала, <i>когда же умрет IE6?</i></p>
```

Несмотря на новые семантические различия, браузеры по-прежнему отображают i курсивом (рис. 1.2). Стиль любого элемента можно изменить и с помощью CSS.

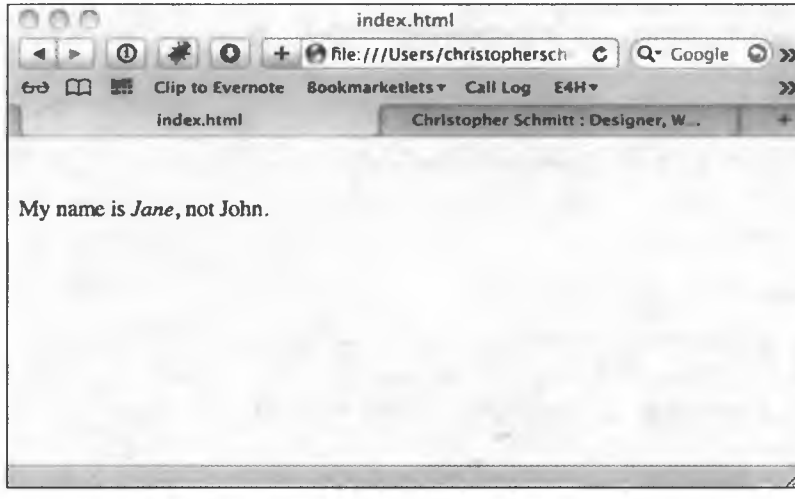

Рис. 1.2. Браузеры выделяют содержимое <i> и курсивом

Дополнительная информация

Статья на HTML5 Doctor «Элементы i, b, em и strong»: http://html5doctor.com/i-b-em-strong-element/.

1.10. Придание тексту значимости

Проблема

Необходимо обозначить в тексте важный фрагмент.

Решение

Поместите важный текст в элемент **strong**:

```
<p>Для совершения действия <strong>требуется регистрация.</strong></p>
```

Обсуждение

В предыдущих версиях HTML **strong** применялся для акцентирования текста. В HTML5 **strong** используется для того, чтобы подчеркнуть важность фрагмента, обычно обозначая оповещение и предупреждение.

Переопределение элемента b. В HTML5 элемент **b** настигла та же участь, что и **i**. Он больше не считается презентационным, а используется для стилистического выделения текста без передачи дополнительного значения.

К примеру, **b** можно применять для выделения ключевых слов, названий или другого текста, который должен отличаться от остального, как, например, начало абзаца:

```
<p><b>Однажды,</b> жил-был один сумасшедший.</p>
```

Разница между **strong** и **b** заключается в семантике, но, как обычно, браузеры одинаково отображают их полужирным шрифтом (рис. 1.3). Конечно же, CSS также позволяет при необходимости изменить стандартный стиль.

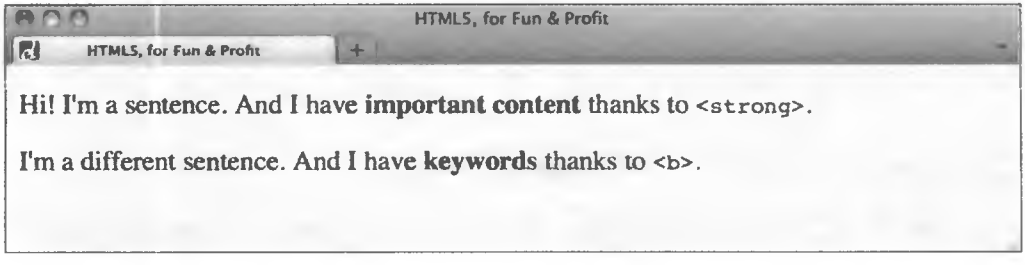

Рис. 1.3. Браузеры выделяют содержимое <b> и полужирным шрифтом

Дополнительная информация

Авторская статья «Красота семантической разметки, часть 2: , <b>, , <i>»: http://ablognotlimited.com/articles/the-beauty-of-semantic-markup-part-2-strong-b-em-i.

1.11. Выделение текста ссылки

Проблема

Необходимо выделить ссылку на результат поиска.

Решение

Поместите текст поискового запроса в элемент **mark**:

```
<p>Поиск по вашему запросу <mark>удивительно</mark>дал 923 результата:</p>
```

Обсуждение

Элемент **mark** — это совершенно новый элемент в HTML5. Пока он в основном используется с поисковыми запросами, но, в соответствии со спецификацией, можно применять его и для текста, который «для справки отмечен или выделен в другом тексте из-за своей значимости».

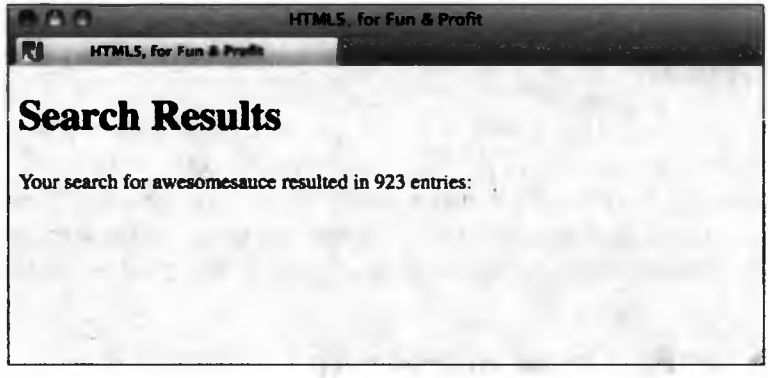

Рис. 1.4. По умолчанию браузеры отображают содержимое <mark> на желтом фоне

Элемент **mark** не придает тексту дополнительного семантического значения и не выделяет его, но современные браузеры (кроме IE до 9 версии) по

умолчанию отображают его содержимое на желтом фоне (рис. 1.4), что вы можете изменить с помощью CSS.

Дополнительная информация

Статья на HTML5 Doctor «Обратите внимание на mark»: http://html5doctor.com/draw-attention-with-mark/.

1.12. Разметка мелким шрифтом

Проблема

Необходимо выделить мелким шрифтом юридическую информацию и сведения об авторских правах.

Решение

Поместите текст, который хотите выделить мелким шрифтом, в элемент **small**:

```
<p><small>Владелец этого блога не несет ответственности за высказанное
    мнение. Пожалуйста, помните, что вы используете его на свой страх
    и риск.</small></p>
<p><small>Авторские права &copy; 2011 Удивительнейший блог. Все права
    защищены.</small></p>
```

Обсуждение

HTML5 переопределяет элемент **small**, ранее считавшийся презентационным. Теперь он применяется для пометок и небольших надписей, например правовой информации, заявлений, сроков, условий и авторских прав. И хотя **small** должен задавать семантику, браузеры по умолчанию отображают его содержимое немного меньшим шрифтом, чем остальной текст (рис. 1.5).

Дополнительная информация

Обсуждение рабочего проекта W3C по поводу семантики текстового уровня HTML5, в том числе элемента **small**: http://www.w3.org/TR/html5/text-level-semantics.html#the-small-element.

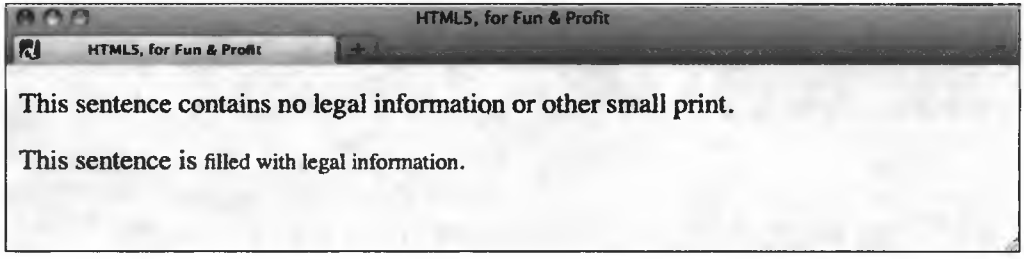

Рис. 1.5. Первое предложение не содержит <small>, а во втором часть текста написана шрифтом немного меньшего размера

1.13. Определение аббревиатур и сокращений

Проблема

Необходимо обеспечить развернутое определение аббревиатур и сокращений.

Решение

Поместите аббревиатуру и сокращение внутрь элемента **abbr** и добавьте определение в качестве значения **title**:

```
<abbr title="Accessible Rich Internet Applications">ARIA</abbr>
```

Обсуждение

В предыдущих версиях HTML для аббревиатур использовались два элемента: **abbr** и **acronym**. Неудивительно, что в веб-мире, постоянно готовом к дискуссии, они вызывали множество споров.

Первая проблема состояла в том, что в IE до версии 7 значения **title** не отображались в виде подсказок (рис. 1.6). Чтобы избежать этого, разработчики просто стали использовать для слов элемент **acronym**, независимо от того, были ли эти слова сокращениями.

Вторая проблема заключалась в различиях **acronym** и **abbr**.

В настоящее время споры утихли, а **acronym** полностью убрали из спецификации HTML5. Причина в том, что все сокращения — это урезанные формы слов или фраз, то есть те же самые аббревиатуры.

Рис. 1.6. По умолчанию в Firefox 5.0 существуют подсказки для <abbr>

Дополнительная информация

Часть действующего стандарта WHATWG, посвященная семантике текста, в том числе элементу **abbr**: http://www.whatwg.org/specs/web-apps/current-work/multipage/text-level-semantics.html#the-abbr-element.

1.14. Добавление ссылок для содержимого блока

Проблема

Необходимо объединить в одной гиперссылке логотип и главный заголовок вашего сайта.

Решение

Поместите изображение с логотипом и заголовок внутрь элемента **a**:

```
<a href="http://html5funprofit.com">
    <h1>HTML5 для работы &amp; отдыха</h1>
    <img src="logo.png" alt="HTML5 для работы и отдыха" />
</a>
```

Обсуждение

Предыдущие версии HTML предполагали использование элемента **a** только для встроенного контента. Таким образом, если у вас были разные элементы, которые требовалось связать с одним ресурсом, необходимо было индивидуально проделать это для каждого:

```
<h1><a href="http://html5funprofit.com">HTML5 для работы &amp;
    отдыха</a></h1>
<a href="http://html5funprofit.com"><img src="logo.png" alt="HTML5 для
    работы и отдыха" /></a>
```

С появлением HTML5 элемент **a** может содержать блочные элементы. И вы по-прежнему можете применять его для встроенного контента. Однако нельзя вложить один элемент **a** в другой.

Возможность упаковки блочных элементов внутрь ссылки — достаточный мотив для использования DOCTYPE HTML5.

Дополнительная информация

Статья на HTML5 Doctor «„Блочные“ ссылки в HTML5»: http://html5doctor.com/block-level-links-in-html-5/.

1.15. Разметка рисунков и подписей

Проблема

Необходимо добавить на страницу иллюстрацию с подписью.

Решение

Поместите изображение в элемент **figure**, а в **figcaption** добавьте подпись в качестве заголовка:

```
<figure>
    <img src="chart.png" alt="Диаграмма повышения удивительности" />
    <figcaption>Увеличение количества удивительного, представленного
        в этом блоге.</figcaption>
</figure>
```

Обсуждение

Элементы **figure** и **figcaption** — новшество HTML5. Несмотря на то что в приведенном выше примере мы ссылаемся на иллюстрацию (**img**), **figure** не ограничивается только этим типом контента. Его можно использовать

с примерами кода, фотографиями, графиками, аудио, данными таблиц и др.

Откуда так много различных типов? Спецификация довольно широко определяет элемент **figure** (http://www.w3.org/TR/html-markup/figure.html) как:

> *единицу контента, возможно, включающую самодостаточный заголовок, на которую ссылаются, как правило, как на единое целое внутри основного потока документа и которая может быть отброшена из него без ущерба для смысла документа.*

Если вы обратили внимание, это определение очень напоминает описание **aside** (http://www.w3.org/TR/html-markup/aside.html). Чтобы узнать, какой элемент вам больше подходит, подумайте, важно ли имеющееся содержимое для понимания окружающего текста. Если оно не так уж и важно, а нужно лишь для связи, то используйте **aside**. В ином случае лучше подойдет **figure**.

Имейте в виду следующее:

- **figcaption** — необязательный элемент и может находиться в любом месте в пределах **figure**;
- тот факт, что пример из этого рецепта содержит изображение, не означает, что все изображения должны использоваться только с **figure**.

Дополнительная информация

Статья на HTML5 Doctor «Simplequiz #4» о **figure**, подписям к ним и тексту **alt**: http://html5doctor.com/html5-simplequiz-4-figures-captions-and-alt-text/.

1.16. Разметка для даты и времени

Проблема

Необходимо перевести дату и время в формат, подходящий для восприятия машиной, но при этом сохранить их вид удобным для чтения человеком.

Решение

Поместите дату и время в понятном человеку виде в элемент **time** и укажите машиночитаемую информацию с помощью атрибута **datetime**:

```
<p>Опубликовано: <time datetime="2011-01-15">15 января 2011</time></p>
```

В зависимости от того, что вы хотите объявить — дату или время, используйте один из вариантов:

- только время:

```
<p>Урок начинается в <time datetime="08:00">8 часов утра</time>.</p>
```

- дата и время (с уточнением часового пояса):

```
<p>Регистрация открыта с <time datetime="2011-01-15T08:00-07:00">8 часов
    утра 15 января 2011, По зимнему времени</time>.</p>
```

- отображать в машиночитаемом формате (дата и время не требуются):

```
<p>Опубликовано: <time>2011-01-15</time></p>
```

Обсуждение

Прежде чем обсудить новый элемент, обратимся к понятию *машинной удобочитаемости*. Оно описывает результат расширения веб-содержимого дополнительной семантической информацией, предназначенной для машин (поисковых систем, пользовательских агентов и т. д.) и придающей тексту больше значимости.

Я не знаю, какие именно машины на данный момент способны отличить datetime от time, но могу предположить, что в скором времени поисковые системы захотят точно отображать чувствительные ко времени результаты новостных запросов.

Теперь поговорим о дате и времени в Интернете. Для обеспечения машинной удобочитаемости дата и время должны быть указаны согласно формату международного стандарта ISO 8601. Для даты используется формат **YYYY-MM-DD**. Для времени суток — формат **hh:mm:ss**. При совместном указании даты и времени необходимо объединить числовые представления с помощью символа **T**: **YYYY-MM-DDThh:mm:ss**.

Итак, вернемся к новому элементу **time**. Как видно из приведенных примеров, можно указать дату, время или дату и время одновременно. Пользователь сможет прочесть содержимое **time**. Эту же информацию машины получат из **datetime** как значение формата ISO 8601.

Стоит упомянуть одно ограничение относительно time: этот элемент нельзя использовать для неточных дат, таких как «август 2011 года». Для даты требуется как минимум день, месяц и год.

Что насчет микроформатов?

На данный момент вас могут заинтересовать *микроформаты* (http://microformats.org), которые основаны на HTML-шаблонах, предназначенных для выражения машиночитаемой семантики. Микроформат hCalendar (http://microformats.org/hCalendar), к примеру, используется для отображения даты и времени.

Хотите больше узнать о микроформатах? Прочтите статью Эмили Льюис «С микроформатами проще» (New Riders, http://microformatsmadesimple.com), где есть большое количество практических примеров, изложенных в доступной форме.

Ни спецификация HTML5 в целом, ни элемент **time** в частности не отвергают применение микроформатов. Вы все еще можете их использовать. На самом деле, чтобы добавить **time** в hCalendar, потребовалось выполнить небольшой семантический трюк. Однако нынешние анализаторы микроформатов не признают элемента **time**, поэтому их нельзя применять вместе.

Это не только трагедия для POSH (см. рецепт 2.1) и приверженцев микроформатов, но и реальность веб-индустрии. Развитие новых технологий оказывает влияние на старые, а дизайнеры и разработчики должны взвесить все за и против и оценить последствия применения того или иного нового элемента.

Дата публикации

Еще одна новая особенность HTML5, связанная с датой, — это атрибут времени **pubdate**. Добавив его, можно определить содержимое **time**, соответствующее дате и времени фактической публикации веб-страницы или статьи (например, в блоге):

```
<p>Опубликовано: <time datetime="2011-01-09" pubdate>9 января,
    2011</time></p>
```

Это позволяет отличить дату публикации от других встречающихся на странице значений **time**.

Дополнительная информация

Обсуждение даты и времени в книге «Погружение в HTML5» Марка Пилгрима: http://diveintohtml5.info/semantics.html#time-element.

1.17. Основы простого разворачивания и сворачивания

Проблема

Необходимо обеспечить возможность разворачивать и сворачивать содержимое страницы, сохраняя фокус на определенной области (если это поддерживается браузером).

Решение

Поместите весь контент внутрь элемента **details** и укажите область, используемую в качестве «фокуса», с помощью **summary**:

```
<details>
    <summary>Предстоящие обсуждения</summary>
    <p>К новому году у нас появится огромное количество статей!</p>
    <ul>
        <li>Понимание алгоритмической структуры</li>
        <li>Когда использовать <code>hgroup</code></li>
        <li>Машинная семантика микроданных</li>
    </ul>
</details>
```

Обсуждение

Этот рецепт описывает способ определения четкой семантической структуры для решения привычных задач, для которых в настоящее время приходится использовать JavaScript. Элемент **summary** указывает, что «управляет» разворачиванием или сворачиванием полного содержимого **details** относительно заданного фокуса (рис. 1.7). Для того чтобы **details** разворачивался по умолчанию (рис. 1.8), необходимо добавить атрибут **open**:

```
<details open>
```

В настоящее время подобное поведение поддерживает только Chrome. В других браузерах для сворачивания/разворачивания контента по-прежнему придется использовать JavaScript, потому что ни один из них не обеспечивает функциональность **details** и **summary**.

Дополнительная информация

Небольшой симпатичный сценарий от Рэми Шарп (Remy Sharp), позволяющий браузерам использовать описанный механизм разворачивания и сворачивания, можно найти по адресу https://gist.github.com/370590.

Рис. 1.7. В Chrome 13 по умолчанию <summary> отображает содержимое <details> свернутым

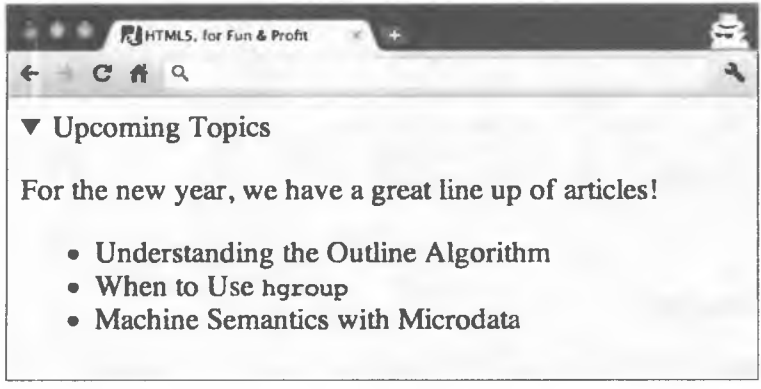

Рис. 1.8. В Chrome 13 по умолчанию <summary> отображает содержимое <details open> развернутым

1.18. Управление нумерацией списков

Проблема

Необходимо упорядочить нумерованный список, по умолчанию начинающийся с 1.

Решение

Воспользуйтесь атрибутами элементов **ol** и **li**. Например, чтобы начать нумерацию списка с 3 (рис. 1.9), добавьте в **ol** атрибут **start** и в качестве его значения установите **3**:

```
<ol start="3">
    <li>Яблоки</li>
    <li>Апельсины</li>
    <li>Бананы</li>
</ol>
```

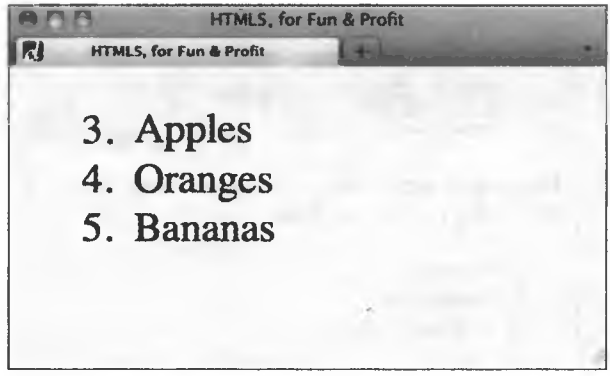

Рис. 1.9. В Firefox 5.0 <ol start="3"> по умолчанию имеет такой вид

Обсуждение

Недавно **start** считался презентационным атрибутом. Но сейчас HTML5 признает его значение в управлении нумерацией и вновь вводит его в обиход. «Упорядочивание» через **start** работает во всех браузерах, в том числе в IE6.

Атрибут reversed

Атрибут **reversed** применяется для управления нумерацией упорядоченных списков и отображает список в обратном порядке (например, для обратного отсчета с 10):

```
<ol reversed>
    <li>Яблоки</li>
    <li>Апельсины</li>
    <li>Бананы</li>
</ol>
```

К сожалению, ни один современный браузер не поддерживает этот атрибут.

Атрибут value

Можно точно указать число **value** для каждого пункта **li**, что пригодится при определении рейтинга, особенно в случае ничейного результата (рис. 1.10):

```
<p>Это рейтинг ваших любимых фруктов, и у нас два претендента на первое
    место!</p>
```

```
<ol>
    <li value="1">Бананы</li>
    <li value="1">Апельсины</li>
    <li value="2">Яблоки</li>
</ol>
```

Все современные браузеры поддерживают для **li** атрибут **value**.

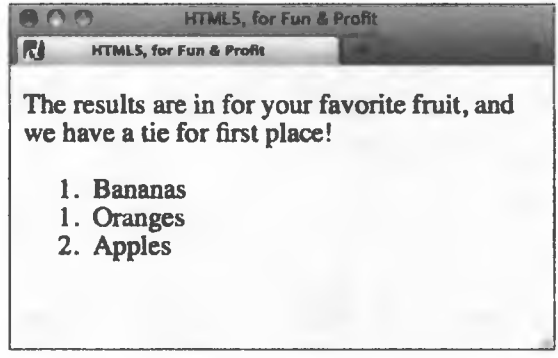

Рис. 1.10. Упорядоченный список <li> с определенными значениями атрибута value

Дополнительная информация

Рабочий проект W3C о группировке контента внутри элемента **ol**: http://www.w3.org/TR/html5/grouping-content.html#the-ol-element.

1.19. Временное скрытие содержимого

Проблема

До тех пор пока пользователь не авторизуется, необходимо скрыть от него часть информации.

Решение

Добавьте к элементу, содержимое которого нужно временно скрыть, атрибут **hidden**:

```
<p hidden>Специальное предложение для участников: используйте код для
    скидки на эту УДИВИТЕЛЬНЕЙШУЮ ВЕЩЬ и сэкономьте 10 % на остальных
    покупках.</p>
```

Обсуждение

Атрибут **hidden** — одно из наиболее спорных дополнений в HTML5. Согласно спецификации, он должен применяться к *несуществующему или недоступному для данного пользователя* контенту, который не должен отображаться браузерами. Объектная модель документа (DOM) и поисковые системы, однако, все еще могут «добраться» до такого содержимого. Иными словами, если информация станет доступной для пользователя, то ее можно будет получить с помощью, например, JavaScript и отобразить, удалив атрибут **hidden**.

Корень зла лежит в непонимании этого атрибута и злоупотреблении им. Он *не* должен использоваться для того, чтобы временно показать/скрыть какой-то текст. Для этого лучше применять вкладки и т. п. Кроме того, существуют уже готовые CSS-решения для выборочного отображения информации, как, например, **display: none** и **visibility: hidden**. Однако атрибут **hidden** обладает большим семантическим значением в сравнении с CSS-альтернативами.

Только не используйте hidden для презентационных целей и не полагайтесь на то, что браузеры действительно скрывают контент. Применяйте hidden лишь для добавления семантики, а CSS — для управления отображением.

Дополнительная информация

Статья Питера Бивирло (Peter Beverloo) «Рассуждения об атрибуте hidden HTML5»: http://peter.sh/2010/06/thoughts-on-the-html5-hidden-attribute/.

1.20. Создание редактируемых частей страницы

Проблема

Необходимо обеспечить пользователям возможность редактировать страницу непосредственно через браузер.

Решение

Добавьте в элемент, содержимое которого может быть отредактировано, атрибут **contenteditable**:

```
<article contenteditable>
    <h2>Не злоупотребляйте <code>section</code>!</h2>
    <p>Основной структурный элемент HTML5 <code>section</code> зачастую
        используется как обычный контейнер для стиля, хотя для этого
        больше подходит <code>div</code>.</p>
</article>
```

Обсуждение

Атрибут **contenteditable** поддерживается современными браузерами, в том числе IE6. Он впервые появился в IE 5.5 и с тех пор поддерживается остальными браузерами. Первоначально атрибут был предназначен для редактирования непосредственно в браузере, то есть выступал в качестве редактора What You See Is What You Get (WYSIWYG) («что видишь, то и получишь»).

Атрибут позволяет пользователям править содержимое и может только сохранить изменения, а не генерировать элементы управления для редактирования, как это бывает в WYSIWYG-редакторах. Для этого придется задействовать JavaScript и локальное хранилище Application Programming Interface (API) (Интерфейс программного приложения) (см. главу 8).

На рис. 1.11 показан хороший пример использования атрибута **contenteditable**. Он демонстрирует, как с учетом CSS-правил или определенной гарнитуры будет выглядеть текст, доступный для редактирования.

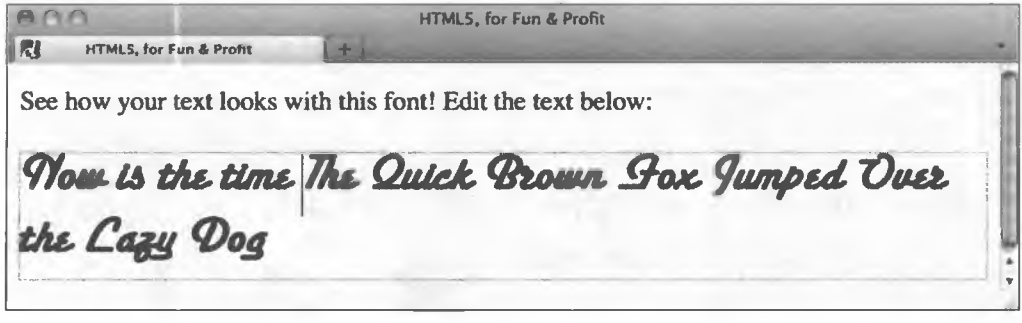

Рис. 1.11. Когда курсор установлен на элементе, обозначенном как contenteditable, Firefox 5.0 отображает его внутри области редактирования, ограниченной пунктиром

Дополнительная информация

Обсуждение рабочего проекта W3C, связанное с пользовательскими взаимодействиями, в том числе с атрибутом **contenteditable**: http://www.w3.org/TR/html5/editing.html#contenteditable.

1.21. Закладываем основы для встроенного перетаскивания

Проблема

Необходимо предоставить пользователям возможность двигать контент на странице.

Решение

Добавьте в элемент, который необходимо перетаскивать, атрибут **draggable**:

```
<h2 draggable>Блочные ссылки</h2>
```

Обсуждение

Атрибут **draggable**, как и **contenteditable**, хорошо поддерживается браузерами. Он также впервые введен в IE (по умолчанию элементы **a** и **img** уже перетаскиваются). Тем не менее обычное добавление к элементу атрибута **draggable** не обеспечивает ему необходимую функциональность. Атрибут лишь позволяет определить элементы, которые можно будет перетаскивать.

Для достижения желаемой функциональности необходимо использовать JavaScript.

1. Сообщите браузеру, где можно будет оставить перетаскиваемый элемент (конец «зоны»).
2. Укажите браузеру, что делать с элементом после того, как он будет отпущен.

Дополнительная информация

Текущий стандарт WHATWG описывает атрибут **draggable** на странице http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html#the-draggable-attribute. Простое понятное руководство по реализации механизма перетаскивания находится по адресу http://html5doctor.com/native-drag-and-drop. Подробнее функциональность механизма перетаскивания описана в рецепте 10.3.

2 Передовые методы разметки

Кристина Хаггинс Рэми

2.0. Введение

Будучи веб-разработчиком, вы наверняка захотите реализовать на своем сайте новые функции HTML5. Тем не менее HTML5 — развивающийся язык, который поддерживается большинством браузеров в лучшем случае частично.

Некоторые вопросы могут испортить ваше приятное впечатление от знакомства с передовыми технологиями HTML5.

- Где найти браузер, поддерживающий большинство HTML5-элементов?
- С помощью каких элементов можно обеспечить кроссбраузерную совместимость HTML5-сайтов?

Из этой главы вы узнаете, как проверить, поддерживает ли браузер функции HTML5 и предоставляет ли он другой способ обеспечить необходимую функциональность. Вы увидите, как можно быстро приступить к работе с шаблоном HTML5 Boilerplate, включающим все новейшие приемы, техники и передовой опыт реализации возможностей языка. Вы узнаете, что требуется для валидации HTML5-страниц и как можно подготовить XHTML/HTML4-документы к предстоящему обновлению до HTML5.

Методы, описанные в этой главе, позволят вам уверенно интегрировать в свои веб-страницы современные веб-технологии HTML5, даже если они опережают возможности браузера.

2.1. Добавление дополнительного семантического значения

Проблема

Необходимо правильно разметить содержимое.

Решение

Используйте *семантическую разметку*, описывающую, из чего состоит содержимое, а не то, как оно должно выглядеть. Это означает, что абзац задается с помощью элемента **p**, названия — с помощью элементов заголовков и т. д.

Обсуждение

Применение описанного способа иногда называют Plain Old Semantic HTML (Старым Добрым Семантическим HTML) (POSH). Семантическая разметка рекомендует избегать для компоновки контента использования таких презентационных элементов, как **font** или **table** (http://microformats.org/wiki/posh).

POSH фокусируется на контенте. Для абзаца используйте **p**. Для заголовка — **h1**. Основная суть заключается в применении элемента, который лучше всего описывает контент.

Чтобы сохранить семантику и валидность кода, нужно ознакомиться с изменениями в HTML5. В новой версии пересмотрены элементы **b** и **i** (см. рецепты 1.9 и 1.10), и полностью исчезли **frame**, **frameset** и **noframes**. Кроме того, исключены презентационные элементы и атрибуты, перечисленные в табл. 2.1.

Таблица 2.1. Список устаревших презентационных функций HTML5

Элементы	Атрибуты
basefont	align
big	background
center	bgcolor
font	border
plaintext	cellpadding
s	cellspacing
strike	height
tt	nowrap
u	valign
	width

Дополнительная информация

Советы и рекомендации по семантической разметке можно прочесть в книге Кристофера Шмитта «CSS: Рецепты программирования» (O'Reilly) или по адресу http://www.w3.org/TR/html5/obsolete.html, где представлен полный список устаревших элементов и атрибутов.

2.2. Выбор стиля разметки

Проблема

Необходимо выбрать синтаксический стиль кода.

Решение

Одно из основных нововведений HTML5 — более свободный синтаксис. Благодаря упрощенному **DOCTYPE** (см. рецепт 1.1) к документу можно применить все стили кода. Будут приемлемы и HTML, и XHTML. Вот некоторые изменения:

- верхний регистр в названиях тегов;
- необязательные кавычки для атрибутов;
- необязательные значения атрибутов;
- необязательные закрывающие теги для пустых элементов.

Итак, вы можете писать код легко и непринужденно, не заботясь о структуре, кавычках и значениях атрибутов, и ваш синтаксис пройдет валидацию.

Стоит отметить, что правильный синтаксис не всегда эквивалентен хорошему. Многие разработчики до сих пор призывают придерживаться определенных «рекомендаций», привносящих ясность в код.

- Пишите названия тегов в нижнем регистре. Верхний регистр читается труднее, поэтому не следует подобным образом отделять код от контента: любой приличный текстовый редактор разметит код вашего HTML-документа цветом, что легко позволит вам увидеть теги.
- Заключайте все значения атрибутов в кавычки. В некоторых ситуациях без кавычек не обойтись (например, существует больше одного значения), но полезная привычка всегда использовать кавычки поможет избежать ошибок и сделает код единообразным.

- Используйте закрывающие теги для всех элементов контента. Если вы будете знать, где заканчивается **div**, ваш код будет более четким. Последовательно выполняя эту рекомендацию, можно достигнуть лучшей читабельности кода, который проще будет поддерживать.

Обсуждение

Независимо от стиля вашего кода, большое значение приобретает единообразие: придерживаясь строгого синтаксиса, вы сможете быстрее обнаружить ошибки в разметке.

Для команды разработчиков основным преимуществом является то, что строгий синтаксис гораздо легче стандартизировать. Если придерживаться общепринятого соглашения, то можно сократить время создания, что особенно актуально для крупных проектов с рассредоточенной командой разработчиков.

В конце концов, в HTML5 не существует правильного или неправильного стиля. Просто придерживайтесь стиля, который поможет вам последовательно разрабатывать код и эффективно устранять ошибки.

Дополнительная информация

Другие рекомендации по поводу синтаксиса вы можете найти на странице http://www.fredzlinkz.com/webcitymaster/html5-syntax-style-recommendations.

2.3. Проверка поддержки HTML5

Проблема

Необходимо разобраться, поддерживает ли браузер или его конкретная версия возможности HTML5.

Решение

На момент написания главы спецификация HTML5 еще не была полностью закончена. Пока она не завершена, браузеры реализуют лишь часть спецификации, то есть в современных обозревателях нет 100 % поддержки HTML5 (так быть не должно).

Из этой книги вы узнаете о том, что существует значительная часть спецификации HTML5, которая поддерживается и может быть использована уже сейчас. Чтобы узнать о том, как HTML5 воспринимается различными браузерами, смотрите страницу http://caniuse.com/#cats=HTML5. Наглядное представление реализации HTML5, а также CSS3 можно найти по адресу http://html5readiness.com, как это показано на рис. 2.1.

Рис. 2.1. Наглядное представление поддержки современными браузерами аудио в простом HTML5

Обсуждение

Конечно, современные браузеры в той или иной степени поддерживают HTML5. Однако никто не заставляет вас применять все возможности новой версии. Вы не обязаны использовать встроенный контент или веб-сокеты, задействуя новые структурные элементы. И наоборот, вам не придется использовать новые структурные элементы, чтобы получить валидный

HTML5-документ. Достаточно лишь определить тип документа (см. рецепт 1.1).

Выберите функции, необходимые вам для работы, и отбросьте остальные. Или поэкспериментируйте со всеми, чтобы увидеть, можете ли вы найти им практическое применение в своих проектах.

Дополнительная информация

Получить наглядное представление о поддержке браузерами HTML5 и CSS3 вы можете на сайте http://html5readiness.com.

2.4. Как заставить Internet Explorer распознавать элементы HTML5?

Проблема

Раньше Internet Explorer не применял к элементам CSS-стили, так как не распознавал их. К сожалению, Internet Explorer (до IE9) до сих пор не воспринимает большинство элементов HTML5. Это означает, что усложняется реализация языка, так как Internet Explorer не различает и CSS-стили, используемые новыми элементами HTML5.

Решение

Неплохое решение этой проблемы предложил Сьорд Висшер (Sjoerd Visscher) (http://intertwingly.net/blog/2008/01/22/Best-Standards-Support#c1201006277). Оно известно как *контейнерный* метод и заключается в создании DOM-элемента с таким же названием, как у элемента HTML5, который необходимо использовать. После появления такого элемента Internet Explorer будет применять к нему CSS.

Предположим, что вы создаете в **body** следующий простой заголовок **header**, устанавливающий навигацию:

```
<header>
    <h1>Загадки Сфинкса</h1>
    <nav>
        <ul>
            <li><a href="index.html">На главную</a></li>
            <li><a href="/about/">О проекте</a></li>
            <li><a href="/blog/">Блог</a></li>
```

```
            <li><a href="/contact/">Как с нами связаться</a></li>
        </ul>
    </nav>
</header>
```

и стили этих элементов:

```
header {
    width: 600px;
    height: 25px;
    background-color: #ccffff;
    border: 2px solid #66cc99;
    padding: 10px;
}

nav {
    width: 100%;
    float: left;
    list-style: none;
}

nav li {
    float: left;
    padding-right: 7px;
}
```

Если открыть эту страницу в Internet Explorer версии 8 и ниже (рис. 2.2), то CSS не работает.

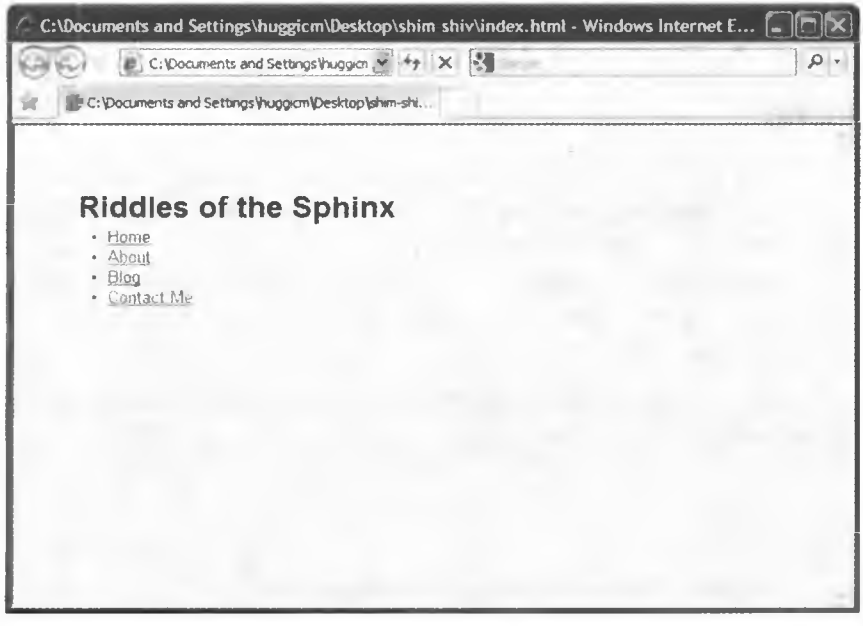

Рис. 2.2. Первоначальное отображение списка

Для того чтобы Internet Explorer распознал элементы, просто добавьте в раздел **head** HTML-документа следующий сценарий (убедитесь, что вы добавили его именно в **head**, так как элементы должны быть объявлены до того, как начнется отображение страницы):

```
<script>
    document.createElement("header");
    document.createElement("nav");
    document.createElement("footer");
</script>
```

Затем объявите как DOM-элементы три HTML5-элемента, которые использованы на странице, — **header**, **nav** и **footer**.

Теперь, если вы откроете эту страницу в Internet Explorer (рис. 2.3), браузер увидит элементы и применит к ним CSS.

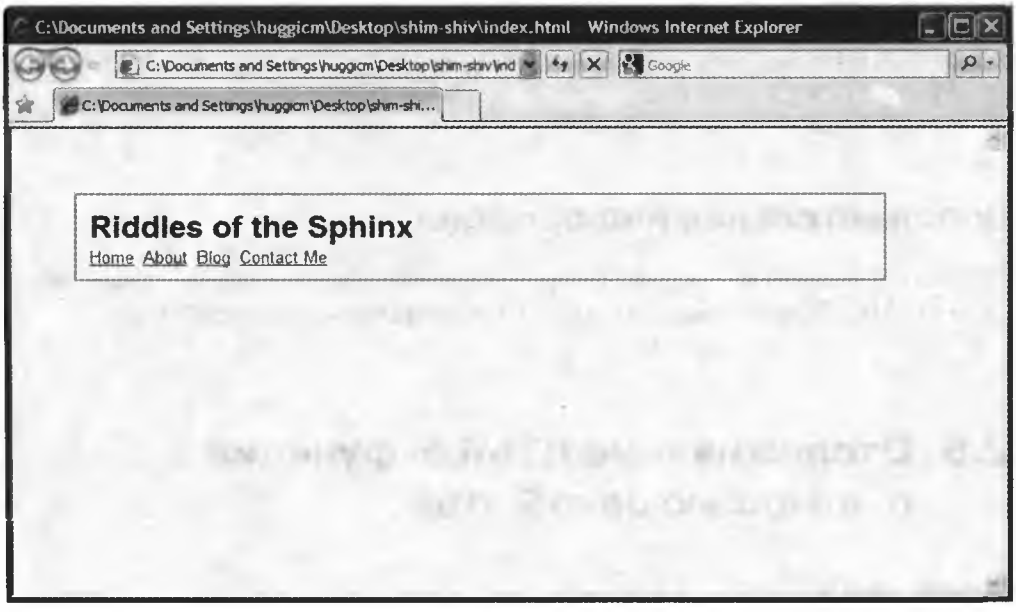

Рис. 2.3. Internet Explorer распознал элементы HTML5

Обсуждение

Хотя такой метод довольно эффективен, создавать DOM-элементы для всех HTML5-элементов на странице может оказаться затруднительным процессом, особенно если многие из них активно применяются.

Веб-разработчик Рэми Шарп создал сценарий, подходящий для всех элементов HTML5. Гораздо удобнее интегрировать его на страницу, вместо того чтобы каждый раз, когда нужно использовать HTML5, вручную прописывать код. Вы можете скачать сценарий по адресу http://remysharp.com/2009/01/07/html5-enabling-script/ и добавить его в раздел **head** своего HTML-кода или брать его из репозитория Google:

```
<!--[if lt IE 9]>
    <script src="http://html5shim.googlecode.com/svn/trunk/html5.js">
    </script>
<![endif]-->
```

Если поместить этот код в тело условного оператора, то сценарий будет вызываться только из браузера Internet Explorer версии 8 и ниже (IE9 широко поддерживает HTML5-элементы и технологии).

Для того чтобы создать элементы, которые должны вести себя как блочные, добавьте свойство display:block в таблицу стилей для новых HTML5-элементов.

Дополнительная информация

В блоге Джона Резига (John Resig) подробно описан контейнерный метод для HTML (Резиг называет его Shiv): http://ejohn.org/blog/html5-shiv/.

2.5. Отслеживание HTML5-функций с помощью JavaScript

Проблема

Необходимо обеспечить работоспособность функций HTML5, которые поддерживаются одними браузерами, но не воспринимаются другими.

Решение

В HTML5 появилось множество интересных функций, которыми вы, будучи дизайнером, наверняка захотите воспользоваться.

К сожалению, в настоящее время HTML5 не в полной мере поддерживается браузерами. Однако хочется реализовать новые технологии и обеспечить их работу в браузерах, которые до сих пор их не воспринимают.

Modernizr (http://modernizr.com) — это библиотека JavaScript, которая может вам помочь. Она определяет работоспособность различных HTML5- и CSS3-функций в браузере пользователя и позволяет создавать запасные варианты, если в настоящее время необходимая функция не поддерживается (рис. 2.4).

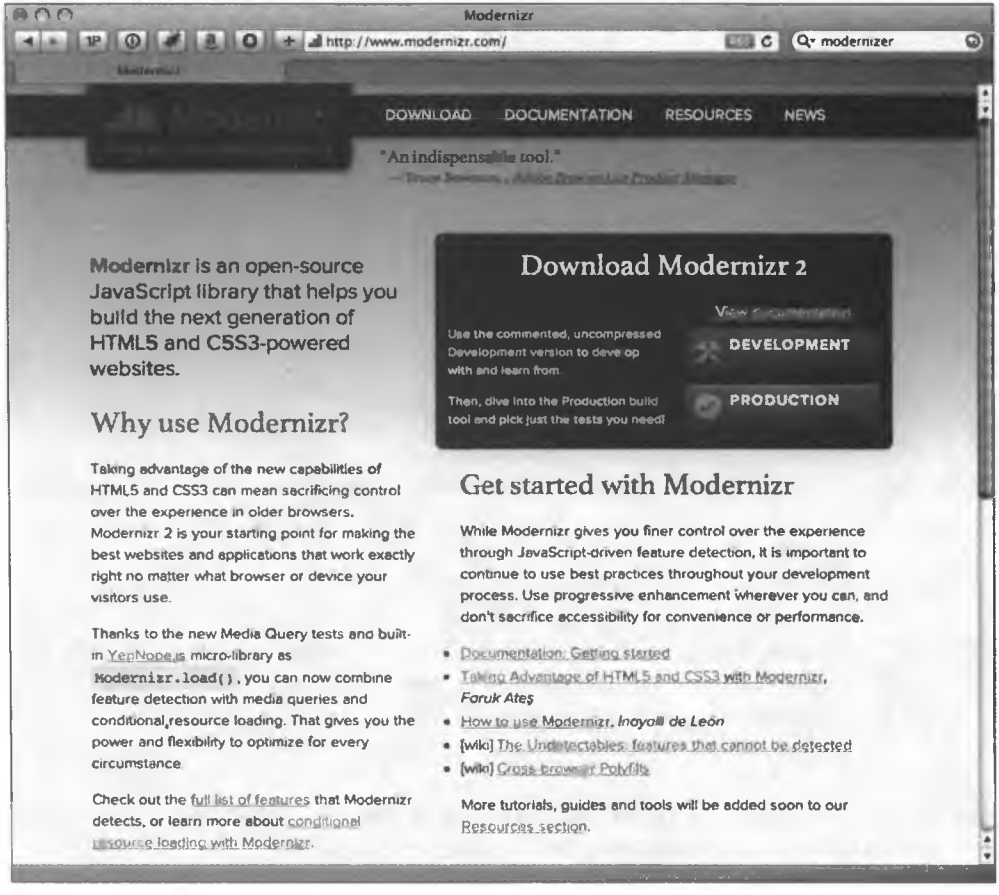

Рис. 2.4. Сайт Modernizr

Использование Modernizr. Для того чтобы увидеть, как работает Modernizr, обратимся к примеру использования библиотеки JavaScript. С помощью элемента **audio** создадим в HTML5 аудиопроигрыватель. Если пользователь

просматривает страницу в браузере, не поддерживающем элемент **audio**, нужно вывести сообщение об ошибке, которое предупредит пользователя о возникшей проблеме.

Если бы мы хотели, чтобы эта страница действительно стала кроссбраузерной, то, вероятно, использовали бы в качестве запасного варианта Flash-аудиопроигрыватель. Тем не менее для этого рецепта нам достаточно простого примера.

Вот основной HTML-код для нашего проигрывателя:

```
<img src="vscover.jpg" alt="Veil &amp; Subdue: Courtship of the Black Sultan"/>
<div id="caption">
    <div id="audio">
        <audio>
            <source src="turnthepage.ogg">
            <source src="turnthepage.mp3">
        </audio>
        <p><input type=button onclick="playPause()" value=" "
            tabindex="0" /> &ldquo;Turn the Page&rdquo; / <em>Veil &amp;
            Subdue</em> / Paul Ramey</p>
    </div>
</div><!--end caption-->
```

Мы добавили изображение с обложкой альбома и элемент **audio**, а также установили панель управления.

Для начала скачаем Modernizr с сайта http://modernizr.com. Нужно добавить ссылку на этот файл в элемент **head** HTML-документа:

```
<script src="js/modernizr-1.6.min.js"></script>
```

Сценарий должен быть добавлен в тело элемента head документа с помощью обычного тега script. Он не может подключаться динамическими методами загрузки сценариев.

Когда этот сценарий запускается, он определяет, поддерживает ли браузер элемент **audio**, а затем делает следующее.

1. Присваивает значение (**true** или **false**) JavaScript-свойству **Modernizr.audio**.
2. Добавляет классы для HTML-тегов. Если браузер поддерживает элемент **audio**, то будет добавлен класс под названием **audio**. Если браузер не поддерживает элемент **audio**, то будет добавлен класс **no-audio**.

Теперь вы можете написать сценарий, указав, что необходимо сделать, если свойство **Modernizr.audio** хранит значение **true** или **false**, и добавить CSS-код, определив стили элементов на те случаи, когда элемент **audio** поддерживается и когда — нет.

Напишем небольшой сценарий (можно поместить его в элемент **head** документа), который работает только в том случае, если элемент **audio** поддерживается:

```
<script>
    if (Modernizr.audio) {
        function playPause() {
            var myAudio = document.getElementsByTagName('audio')[0];
            if(myAudio.paused)
                myAudio.play();
            else
                myAudio.pause();
        }
    }
</script>
```

Этот сценарий приводит в действие пользовательскую панель управления, внедренную на страницу. Она будет активна, только если **Modernizr.audio** принимает значение **true** (и, следовательно, не расходует ресурсы браузера, неподдерживающего элемент **audio**).

Теперь было бы неплохо иметь возможность вообще избавиться от панели управления в том случае, если браузер не поддерживает элемент **audio**. Это можно легко сделать с помощью Modernizr, воспользовавшись классом для HTML-тегов:

```
.no-audio #audio {
    display: none;
}

.audio #audio input[type="button"] {
    width: 45px;
    height: 29px;
    background: url(button.png);
    border: 0px solid #000;
    float: left;
    margin-right: 10px;
}
```

Когда элемент **audio** не поддерживается, Modernizr добавляет в раздел **header** класс **no-audio**. Таким образом, можно выбрать для этого класса CSS-стиль и установить параметры отображения для любого элемента, который нужно спрятать. Это мог бы быть раздел **div** под названием **"audio"**, содержащий панель управления.

Если элемент **audio** поддерживается, то Modernizr добавляет в раздел **header** класс **audio**. В этом случае можно выбрать для класса любой желаемый стиль панели управления.

Следующий шаг — создание сообщения об ошибке на тот случай, когда элемент **audio** не поддерживается. Во-первых, необходимо добавить в HTML поле для вывода этой ошибки, а затем поместить предупреждение в раздел **div "audio"**:

```
<div id="error">
    <p>&ldquo;Turn the Page&rdquo; / <em>Veil &amp; Subdue</em> / Paul
        Ramey</p>
    <div id="error-box">
        <p>Ужасно! Аудио на этой странице не поддерживается браузером.
            Не желаете установить последнюю версию браузера?</p>
    </div>
</div>
```

Используя указанный выше основной метод, мы можем скрыть раздел **div**, если элемент **audio** поддерживается, и применить его стиль в ином случае:

```
.audio #error {
    display: none;
}

.no-audio #error-box {
    background-color: #ffffcc;
    padding: 1px 10px;
    color: #000;
    border: 5px solid #ffff66;
}

.no-audio #error-box a {
    color: #333;
}
```

На рис. 2.5 показан вариант, когда элемент **audio** поддерживается браузером, в то время как на рис. 2.6 приведен пример, когда элемент **audio** не поддерживается.

На сегодняшний день библиотека Modernizr способна проверить более 20 функций HTML5 и CSS3, а также позволяет использовать контейнерный метод, рассмотренный в рецепте 2.4. Документацию к Modernizr можно найти по адресу http://modernizr.com/docs/; она содержит образцы кода для каждой поддерживаемой функции.

Более подробную информацию и рецепты использования аудио в HTML5 ищите в главе 5.

Обсуждение

Modernizr — это не только инструмент, позволяющий находить проблемы и избегать их: традиционно «UA-перехват», или «перехват браузера», применялся, чтобы определить, с каким браузером работает пользователь. Затем программист мог написать код, ориентированный на конкретный браузер, для индивидуализации страницы.

Рис. 2.5. Панель управления аудио в браузере, поддерживающем элемент audio

Однако этот метод не надежен в нескольких случаях. Во-первых, браузер может подделать строку UA — значение, которое перехватывается у него для определения его типа. Кроме того, пользователи могут отключить некоторые функции браузера, так что даже если вы сумеете правильно определить его тип, то все равно не будете наверняка знать, что именно поддерживается.

Если же пользователь запускает новый браузер, новую версию браузера или альтернативный браузер, который вы, возможно, не догадались проверить, ваш код вообще может не определить тип обозревателя. Не говоря уже о том, что на *ваши* плечи ложится обязанность следить за тем, какие функции в каких браузерах выполняются.

Рис. 2.6. Так выглядит страница, когда элемент audio не поддерживается браузером

Modernizr позволяет избежать подобных ошибок, так как определяет, какие именно *функции* поддерживаются, а не просто пытается убедиться, что браузер пользователя полностью показал страницу. Воспользуйтесь этим довольно надежным инструментом, если хотите максимально подстраиваться под посетителей своего сайта.

Сейчас в Modernizr появилась возможность выбора версии (http://www.modernizr.com/download/), в которой генерируются инструменты проверки исключительно для HTML5-функций, необходимых для ваших сайтов или веб-приложений.

Дополнительная информация

Документация на официальном сайте Modernizr: http://www.modernizr.com/docs/.

2.6. Использование HTML5 Boilerplate

Проблема

Необходимо приступить к работе с HTML5, используя проверенный шаблон.

Решение

Использование шаблона HTML5 Boilerplate Пола Айриша (Paul Irish) — отправная точка для сайта (рис. 2.7). Boilerplate включает в себя HTML5-шаблон в комплекте с организованной структурой папок и CSS-файлом. Кроме того, он основан на современном опыте программирования, позволяет исправлять ошибки браузера, а также содержит популярные библиотеки JavaScript.

Шаблон Boilerplate представляет собой результат трехлетних исследований передовых методов кодирования, и вы можете применять его в качестве основы для своего сайта. На самом деле HTML5 Boilerplate содержит так много, что пора поговорить о нем подробнее. Итак, рассмотрим основные элементы.

- Файл **index.html**, содержащий:
  - валидную разметку HTML5;
  - ссылки на Modernizr и JQuery библиотеки JavaScript (включенные в шаблон);
  - условные операторы с классами для тегов **body**, необходимыми для отображения страницы во всех версиях Internet Explorer (вы можете легко создать стиль, ориентированный на IE6, IE7);
  - основную структуру тега **body**;
  - сценарий IE6 для PNG-файлов;
  - сценарий Google analytics.
- CSS-файл, включающий в себя:
  - код для полного сброса стилизации;
  - правила для нормализации шрифтов (ваши шрифты будут иметь одинаковый размер во всех браузерах);

- основные стили для списков, таблиц и т. д.;
- несколько несемантических классов (для замены изображений и скрытия элементов);
- стили для печати;
- прокрутку для мобильных браузеров и стили, зависящие от ориентации (к примеру, в IPad страницы отображаются горизонтально, а не вертикально).

Рис. 2.7. Сайт HTML5 Boilerplate

- Favicon (значок для Избранного) и Apple touch icon (значок для приложений Apple) (которые вы, возможно, захотите настроить).
- Файлы конфигурации веб-сервера.

Работа с HTML5 Boilerplate. В первую очередь скачайте с сайта https://github.com/paulirish/html5-boilerplate последнюю версию HTML5 Boilerplate.

Разархивируйте файлы и поместите их в корневой каталог своего веб-проекта. В документации сказано, что для начала необходимо удалить нескольких файлов:

- демо-файлы (находятся в папке под названием Demo);
- код профайлера (расположен в конце файла **index.html** и начинается с комментария **yui profiler and profileviewer — remove for production**).

Используйте файл **index.html** в качестве шаблона для всех своих HTML-документов. Включите в него главные элементы и контейнеры **div** или начните с нуля и создавайте свои собственные. Поскольку шаблон, созданный Полом, — это отличное начало, не стесняйтесь применять его в качестве отправной точки.

Кроме того, вы можете задавать информацию и в заголовке документа **head** (название, автор и т. д.).

В конце HTML-документа добавьте персональный код отслеживания своего сайта в Google Analytics. Если же вы не используете Google Analytics, то удалите этот раздел.

Теперь вам необходимо настроить файл **style.css**. За его основу можно взять CSS-файл из шаблона. Он начинается с комментария:

```
/*  Primary Styles
    Author:
*/
```

После того как вы добавили основной CSS-файл, смело задавайте стили, определяющие ориентацию, и стили для мобильных браузеров. Вы найдете их в разделе, начинающемся с комментария:

```
/*
 * Media queries for responsive design
 * These follow after primary styles so they will successfully override.
 */
```

В завершение настройте значок для Избранного и значок для приложений Apple. Это можно легко сделать, заменив в шаблоне текущие файлы значков.

Обсуждение

Довольно просто создать основу для HTML5-сайта, используя Boilerplate HTML5. Но прежде чем получить от шаблона максимальную отдачу, придется изучить документацию, чтобы понять, что же происходит с кодом.

Если вы не понимаете, как шаблон включает в HTML классы (к примеру, IE-классы **body** или класс для замены изображения), то не сможете применять на своем сайте необходимые для этого фрагменты CSS. Точно так же, если вы не знаете, как реализовать исправление IE6 для PNG-файлов, входящее в шаблон, этот сценарий окажется бесполезным.

Кроме того, в шаблоне могут встретиться неиспользуемые вами функции, от которых вы захотите избавиться. Вы можете удалить сценарий для PNG-файлов, если вас не волнует поддержка IE6, или удалить файл **.htaccess**, если не пользуетесь сервером Apache. Зная, что входит в шаблон, вы легко сможете отбросить все лишнее и оставить необходимое.

Это не значит, что нужно проводить уйму времени, закопавшись в документацию, чтобы понять все особенности внутреннего устройства шаблона. Вы и сами можете создать такой же шаблон, если захотите.

Тем не менее предполагается, что вы должны иметь некоторое представление о составляющих шаблона. Если он содержит что-то на ваш взгляд полезное для сайта, то можете изучить это подробнее.

Дополнительная информация

Официальное руководство по HTML5 Boilerplate: http://net.tutsplus.com/tutorials/html-css-techniques/the-official-guide-to-html5-boilerplate/, а также документация к HTML5 Boilerplate: http://html5boilerplate.com/docs/.

2.7. Валидация HTML5

Проблема

Необходимо найти и исправить ошибки в HTML-коде.

Решение

Валидация кода — одно из ключевых понятий в веб-разработке. Это первый шаг к устранению ошибок кодирования. Валидация не гарантирует, что веб-страница будет выглядеть и вести себя так, как вы того хотите. Она гарантирует, что любое непредвиденное поведение страницы никак не будет связано с несоответствующим тегом или неправильным CSS-селектором.

Зачем нужна валидация? К счастью (или к сожалению, в зависимости от вашего отношения к веб-стандартам), большинство браузеров пытаются компенсировать недостатки кодирования. Таким образом, даже ужасно скомпонованные, несоответствующие и неправильно вложенные теги, вероятно, будут неплохо выглядеть в большинстве браузеров. Тем не менее различные браузеры по-разному обрабатывают ошибки. Это значит, что, если код неверен, браузер может попытаться исправить ошибку самым неожиданным образом.

Мобильные браузеры менее снисходительны к неверному коду (они, в отличие от настольных собратьев, просто не наделены возможностями, позволяющими компенсировать ошибки). Таким образом, неправильная страница, которая успешно отображается в обычном браузере, на мобильном телефоне может дать сбой. В свете современных веб-технологий растет важность валидации кода и устранения ошибок. При этом вы делаете еще один шаг вперед к обеспечению корректного отображения вашей веб-страницы различными браузерами.

К счастью, валидация HTML5 не сильно отличается от валидации XHTML/HTML4. Вы увидите, что создать HTML5-документ в действительности гораздо проще, так как HTML5 имеет куда более мягкую спецификацию, чем его предшественники. Если вы хотите использовать HTML-теги — вперед, прежний HTML еще валиден. Если вы слишком ленивы для того, чтобы закрывать теги, не вините себя. Валидатор не будет бить вас, он предоставит эту возможность вашим коллегам.

Итак, перейдем к процедуре валидации кода. В настоящее время можно использовать любой из двух популярных онлайн-валидаторов:

- единый валидатор W3C Unicorn (рис. 2.8), выполняющий различные проверки и валидацию HTML5 и CSS3: http://validator.w3.org/unicorn/;
- HTML5-валидатор Validator.nu: http://html5.validator.nu/.

Для проверки кода можно ввести URL-адрес, загрузить HTML-файл или просто вставить в валидатор содержимое файла. Когда вы нажимаете кнопку Check (Проверить), валидатор проверяет, в порядке ли этот HTML-код (и CSS, если вы также его проверяете), и выводит список ошибок с номером строки, как показано на рис. 2.9.

Как правило, необходимо несколько раз пропустить страницу через валидатор: одна синтаксическая ошибка (например, неправильно закрытый тег) может спровоцировать каскад ошибок, исчезающих после исправления начальной, так что перепроверяйте свой документ по мере исправления.

В настоящее время W3C валидатор Unicorn по умолчанию проверяет CSS 2.1, но вы можете настроить эту функцию, также включив CSS3.

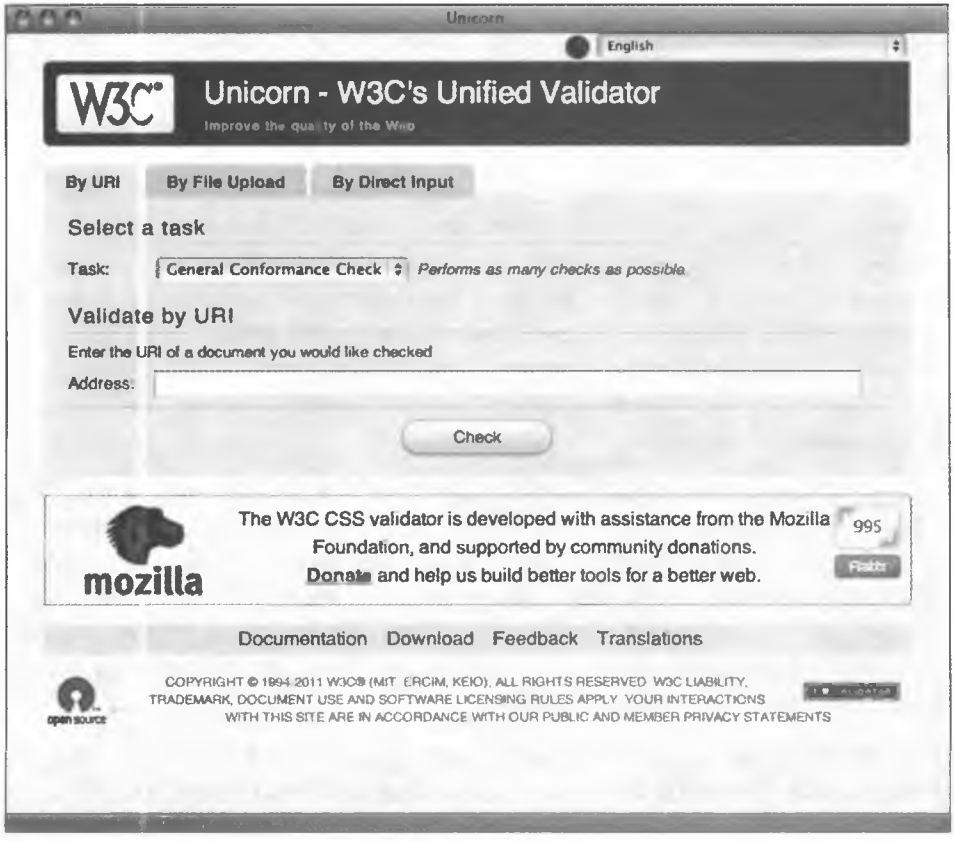

Рис. 2.8. Единый валидатор под названием Unicorn

Обсуждение

Валидация — еще не залог хорошего HTML-кода. Бывают случаи, когда разработчик умышленно включает в HTML-документ неверный код. Это может быть код, который не проходит валидацию, или новое CSS-правило, которое еще недостаточно широко поддерживается различными браузерами (и, следовательно, не принимается современными валидаторами). Все это может стать веской причиной для того, чтобы веб-документ не прошел валидацию. И хотя можно поспорить, стоит ли использовать какой-либо код или новую возможность CSS, умышленное включение в код неверных фрагментов все еще встречается.

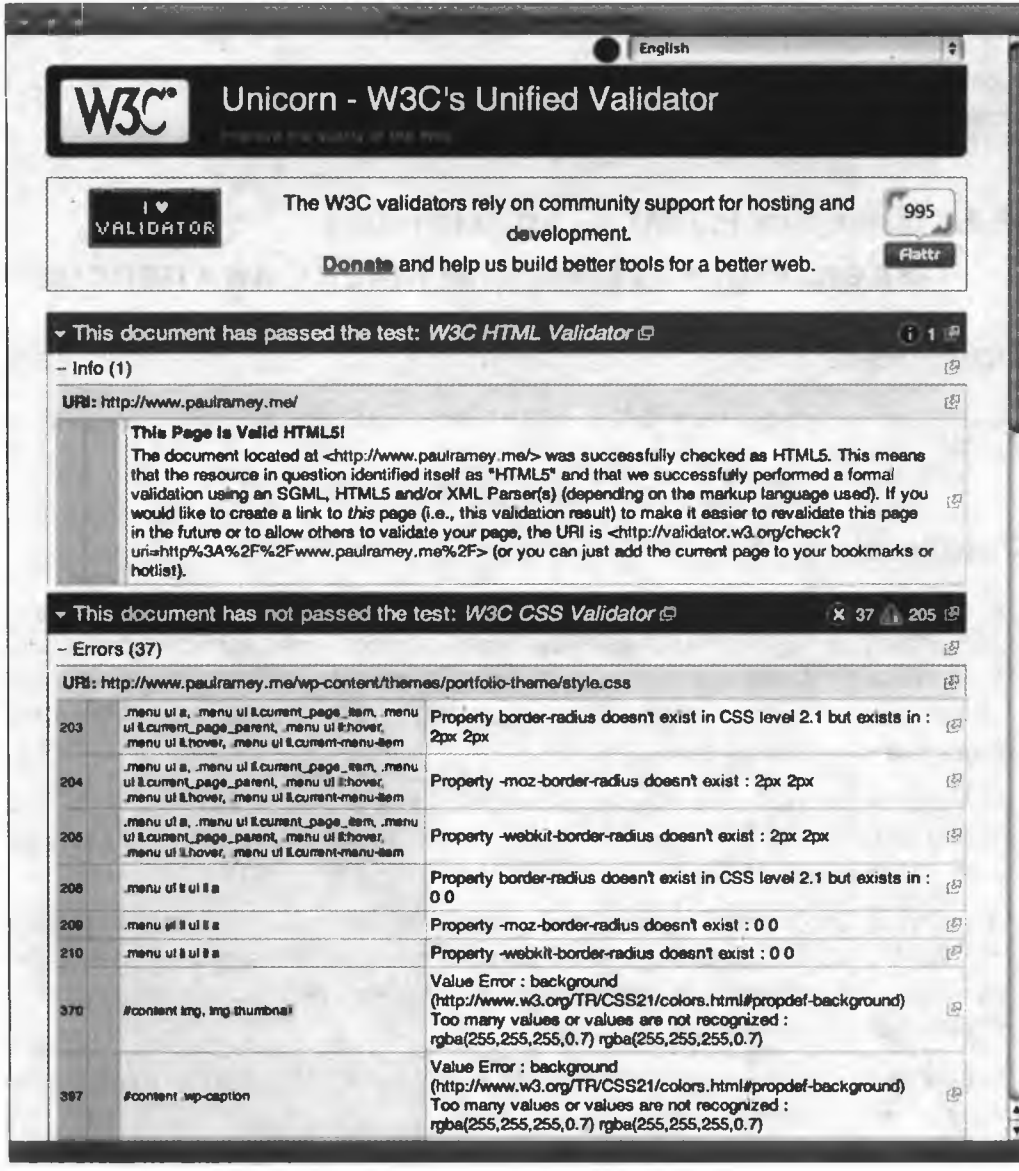

Рис. 2.9. Результаты валидации

Валидация должна рассматриваться как один из инструментов для веб-разработки, а не как решающий голос относительно того, правильно ли кодируется ваша страница. Это отличный способ найти в коде случайные ошибки, к тому же валидный HTML5-документ, как правило, лучше, чем не валидный.

Дополнительная информация

Документация по двум рассмотренным сервисам представлена по адресам http://code.w3.org/unicorn и http://about.validator.nu/.

2.8. Замена HTML5-элементов идентификаторами и именами классов

Проблема

Вы еще не используете HTML5, но вам необходимо подготовить код своего сайта к переходу на новую версию.

Решение

У вас могут быть законные основания, по которым вы еще не готовы использовать HTML5. Возможно, вы беспокоитесь о поддержке браузеров. Может быть, вы не желаете выполнять множество сценариев и настроек, необходимых для того, чтобы сделать HTML5-сайт совместимым со всеми современными браузерами.

Однако вы уже придерживаетесь семантической разметки HTML5 и хотите, чтобы ваш сайт на XHTML/HTML4 работал во всех браузерах и был готов к переходу на HTML5.

К счастью, можно довольно легко воспроизвести семантическую структуру HTML5-сайта, используя контейнер **div** для идентификаторов и имен классов. Рассмотрим, как это сделать. Создадим простую страницу блога, как показано на рис. 2.10.

Глядя на этот сайт, можно выделить несколько ключевых структурных элементов:

- заголовок **body**, содержащий название блога и панель навигации;
- раздел, включающий две записи блога;
- нижний колонтитул.

Если бы мы писали этот сайт на HTML5, то разметили бы его с помощью элементов **header** и **footer**, **nav** и **article**. Однако мы не собираемся так размечать его в XHTML/HTML4, а вместо этого хотим повторить структуру и имена для HTML5, насколько это возможно сделать с помощью **div**, идентификаторов и имен классов.

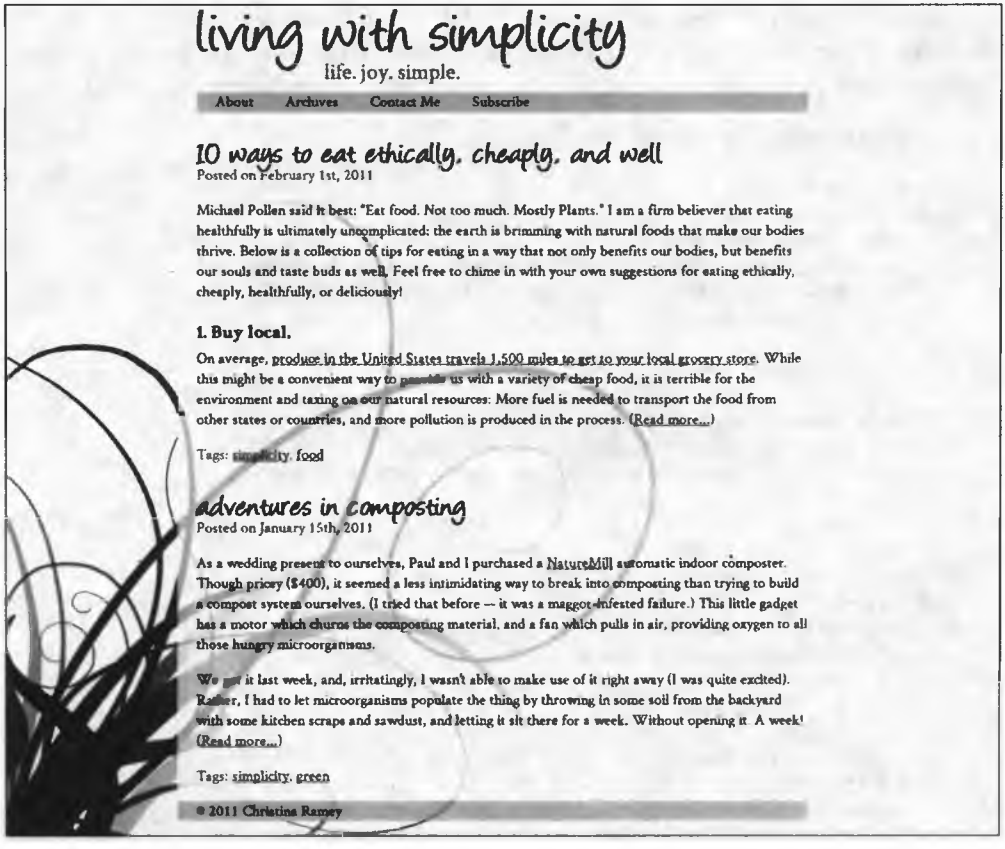

Рис. 2.10. Пример веб-страницы

Во-первых, поработаем с заголовком:

```
<div id="header">
    <h1>living with simplicity</h1>
    <div id="tagline">
        life. joy. simple.
    </div>
    <div class="nav">
        <ul>
            <li><a href="about/">О проекте</a></li>
            <li><a href="archives/">Архивы</a></li>
            <li><a href="contact/">Как с нами связаться</a></li>
            <li><a href="subscribe/">Подписаться</a></li>
        </ul>
    </div><!--end nav-->
</div><!--end header-->
```

В результате все это будет мало отличаться от того, как страница была написана изначально. Но обратите внимание, что в **div** мы повторили имена

HTML5-элементов "**header**" и "**nav**". В HTML5 **header** должен включать в себя тег заголовка (у нас есть тег заголовка **h1**), а также может хранить элемент **nav** (поэтому добавим в заголовок навигационную панель).

Далее разметим наши записи в блоге:

```
<div class="posts">
    <div class="article">
        <div class="header">
            <h1>10 способов питаться правильно, дешево и хорошо</h1>
            <div class="meta">Опубликовано <span class="date">1 февраля
                2011</span>
            </div>
        </div><!--end header-->
        [Post Content]
        <div class="footer">
            <div class="meta">Теги: <a href="tags/simplicity"
                rel="tags"> простота</a>, <a href="tags/food">
                еда</a>
            </div>
        </div><!--end footer-->
    </div><!--end article-->
    <div class="article">
        <div class="header">
            <h1>приключения в компосте</h1>
            <div class="meta">Опубликовано <span class="date">15 января
                2011</span>
            </div>
        </div><!--end header-->
        [Post Content]
        <div class="footer">
            <div class="meta">Tags: <a href="tags/simplicity"
                rel="tags">простота</a>, <a href="tags/green/"
                rel="tags">зелень</a>
            </div>
        </div><!--end footer-->
    </div><!--end article-->
</div><!--end posts-->
```

Для начала мы поместили наши записи в контейнер **div** с надписью "**posts**". Вы можете назвать его "**section**" (как соответствующий HTML5-элемент) – в конце концов, записи в блоге сами по себе похожи на отдельный раздел. Однако помните, что элемент **section** не может использоваться, как раньше использовался **div**, поэтому уместно применять "**section**" только в том случае, если он содержит хотя бы один тег заголовка (то есть представляет собой структурный уровень HTML-документа). В этом **div** нет тега заголовка, поэтому мы дали ему альтернативное семантическое имя.

Затем мы поместили каждую запись в контейнер **div** с названием "**article**". В HTML5 он соответствует элементу **article**. Каждый такой **div**, в свою очередь, содержит верхний и нижний колонтитулы (которые будут совпадать с HTML5-элементами **header** и **footer**).

Интересно, что каждый заголовок у нас хранится в элементе **h1**. До появления HTML5 наличие нескольких **h1** в XHTML/HTML-документе не приветствовалось. Для того чтобы ваша страница была правильно представлена в DOM, заголовки должны были следовать в естественном порядке: элементы **h3** — за **h2**, **h2** и **h1** — после элемента с заголовком страницы. С появлением HTML5 необходимость в этом отпала: браузер понимает элемент **article** семантически, поэтому тот может ранжироваться (и структурироваться) иначе, чем, скажем, элемент **header**.

Таким образом, вам больше не приходится полагаться на теги заголовков при структурировании информации в документе — эта задача решается с помощью новых семантических элементов.

Каждый ваш элемент (например, **header** или **article**) может начинаться со своего элемента **h1**. Хотя применение новых элементов в XHTML/HTML4-документе может показаться странным, тем не менее это облегчит переход на HTML5.

Наконец, мы закрываем страницу элементом **div** под названием **"footer"**, соответствующим в HTML5 элементу **footer**:

```
<div id="footer"> <p>&copy; 2011 Кристина Рэми</p> </div>
```

Теперь, когда мы готовы перейти на HTML5, нам не составит труда заменить все элементы **div** соответствующими элементами HTML5, что позволит избежать головной боли из-за любых глобальных преобразований на странице.

Обсуждение

Нет ничего сложного в замене HTML5-элементов в XHTML/HTML4-документах с помощью контейнеров **div**. На самом деле вам практически ничего не придется исправлять в своем коде. Однако, как только вы определили идентификаторы и имена классов для соответствующих HTML5-элементов, продолжайте придерживаться выбранного порядка именования (особенно, если работаете с несколькими страницами). В результате, когда придет время перевести страницы в HTML5, вам не придется вспоминать, что в одном документе вы назвали **header** как **"header"**, а в другом — как **"main-header"**. Кроме того, это может облегчить быстрый поиск и замену в нескольких документах с помощью текстового редактора.

Дополнительная информация

Шпаргалка по идентификаторам и именам классов HTML5 от Оли Студхолма (Oli Studholme): http://oli.jp/2008/html5-class-cheatsheet/.

3 Формы

Кимберли Блессинг

3.0. Введение

Формы — это основной инструмент любого веб-приложения. Когда вы обновляете свой статус в Facebook, оплачиваете счета в Интернете или удаленно программируете видеорегистратор, все это волшебство становится возможным благодаря формам.

В настоящее время для создания подобных чудес используются некоторые интересные и замысловатые функции HTML и JavaScript. С большей частью работы легко справляется HTML5, и даже если браузеры пока не принимают новые возможности HTML5-форм, можно легко применять их в качестве структурных блоков, поддерживаемых JavaScript.

Для получения поля формы в языке HTML5 предусмотрены новые элементы и новые значения атрибутов элемента **input**, а также несколько атрибутов, позволяющих избежать привязки к JavaScript. Однако еще не все браузеры поддерживают новые функции, поэтому иногда для обеспечения кроссбраузерности приходится задействовать JavaScript.

3.1. Отображение поискового поля ввода

Проблема

Необходимо предоставить пользователю поле для поиска.

Решение

Используйте элемент **input** с атрибутом **type**, имеющим HTML5-значение **search**:

```
<form>
    <p><label>Найти <input type="search" name="query"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Поле ввода типа **search** отображается как текстовое поле ввода, которое визуально может отличаться от обычного текстового поля в зависимости от стиля платформы.

Например, Safari на Mac OS отображает поле поиска со скругленными (а не прямыми) углами (рис. 3.1). Safari и Chrome показывают в поле поиска значок, позволяющий удалить текущее значение (рис. 3.2).

Рис. 3.1. Пустое поле поиска в Safari 5

Некоторые пользовательские агенты вместо поля ввода типа **search** будут отображать простое текстовое поле ввода, так что наличие поддержки определяется не только по внешнему виду.

Рис. 3.2. Пустое поле поиска в Chrome 12

Проверка поддержки. Если вам необходимо достоверно убедиться, что браузер поддерживает поле ввода типа **search** (см. список совместимых браузеров в табл. 3.1), загрузите для проверки страницу Майка Тейлора (Mike Taylor): http://www.miketaylr.com/code/input-type-attr.html. Здесь вы получите не только сведения о поддержке поля ввода типа **search**, но и данные о значениях атрибутов элемента **input**.

Таблица 3.1. Браузеры, поддерживающие поле ввода типа search

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	4.0+	10+	4.0+	11.0+	✓	✓

Если пользовательский агент не поддерживает поле ввода типа **search**, оно будет отображаться как текстовое поле ввода. Зачем тогда нужен специальный тип для поискового поля ввода, если оно будет визуально отличаться лишь для некоторых пользовательских агентов? Существует несколько ответов на этот вопрос.

Во-первых, визуализация в браузере стилей, напоминающих стили операционной системы, служит для повышения удобства работы пользователей. Во-вторых, поле ввода, обозначенное как поле поиска, позволит вспомогательным устройствам точнее передать сигнал пользователя в поле или форму.

Наконец, поле ввода типа **search** при употреблении в формах имеет большее семантическое значение, а также увеличивает возможности CSS по сравнению с использованием классов или идентификаторов. Следующий фрагмент CSS-кода применяется для задания скругленных углов поля поиска во всех пользовательских агентах, поддерживающих свойство **border-radius**:

```
input[type="search"] {
    border-radius: 10px;
}
```

Дополнительная информация

Дополнительные сведения о поле поиска в браузерах WebKit, в том числе Chrome и Safari, можно получить по адресу http://css-tricks.com/webkit-html5-search-inputs/.

3.2. Поле ввода контактной информации

Проблема

Необходимо предоставить пользователю форму для ввода контактной информации, например адреса электронной почты, URL или номера телефона.

Решение

Используйте элемент **input** с атрибутом **type**, имеющим HTML5-значения **email**, **url** и **tel**:

```
<form>
    <fieldset>
        <legend>Контактная информация</legend>
        <p><label>E-mail <input type="email" name="email"></label></p>
        <p><label>Сайт <input type="url" name="website"></label></p>
        <p><label>Телефон <input type="tel" name="phone"></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Сколько раз вы создавали текстовые поля для ввода адресов электронной почты, а затем приходилось писать сценарий JavaScript для проверки введенных данных? С новыми полями типа **email** и **url** вам больше не придется этого делать. Поле такого типа позволяет вводить только адрес электронной почты или абсолютный URL соответственно. Если пользователь набирает недопустимое значение, то пользовательский агент предупреждает его об этом и не позволяет отправить данные формы (см. примеры этого в различных браузерах на рис. 3.3–3.6).

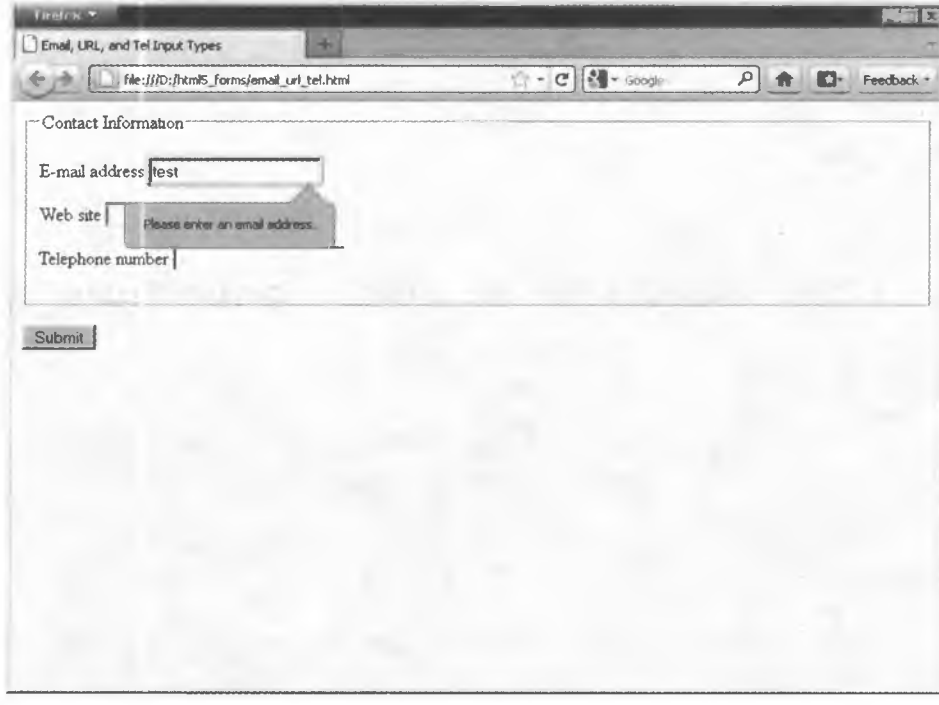

Рис. 3.3. Сообщение об ошибке в поле ввода типа email в Firefox 4

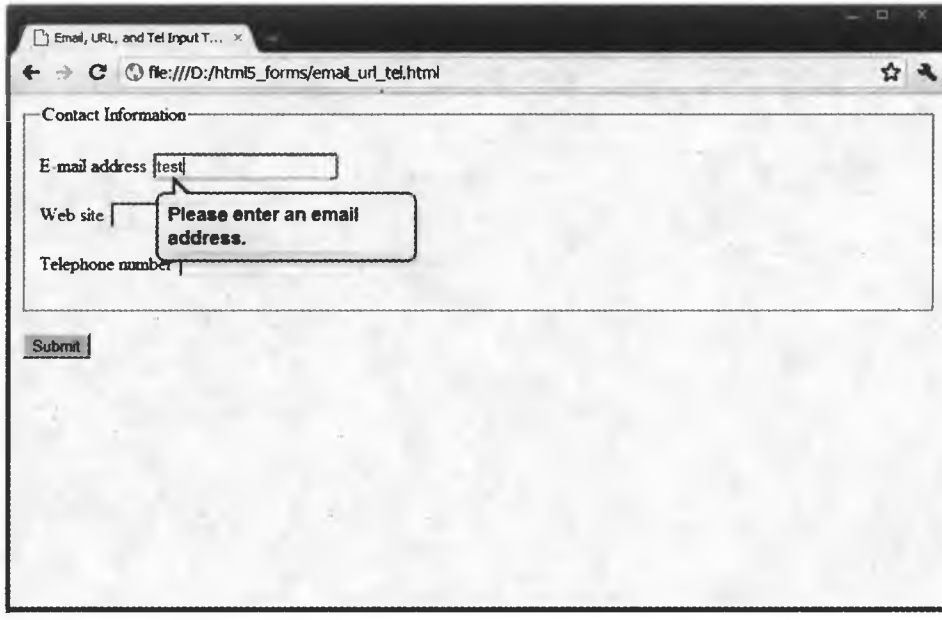

Рис. 3.4. Сообщение об ошибке в поле ввода типа email в Chrome 12

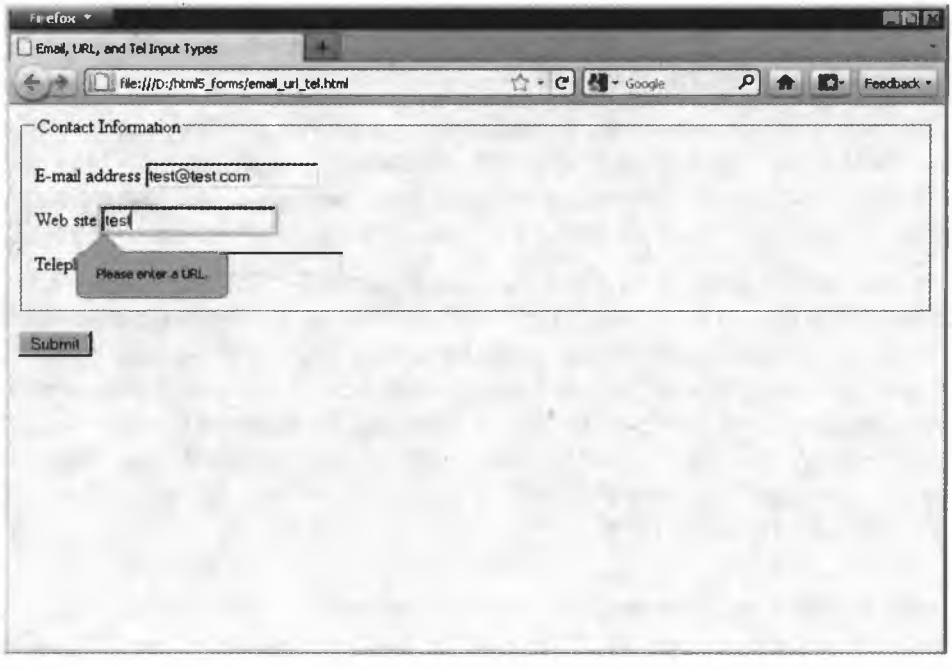

Рис. 3.5. Сообщение об ошибке в поле ввода типа url в Firefox 4

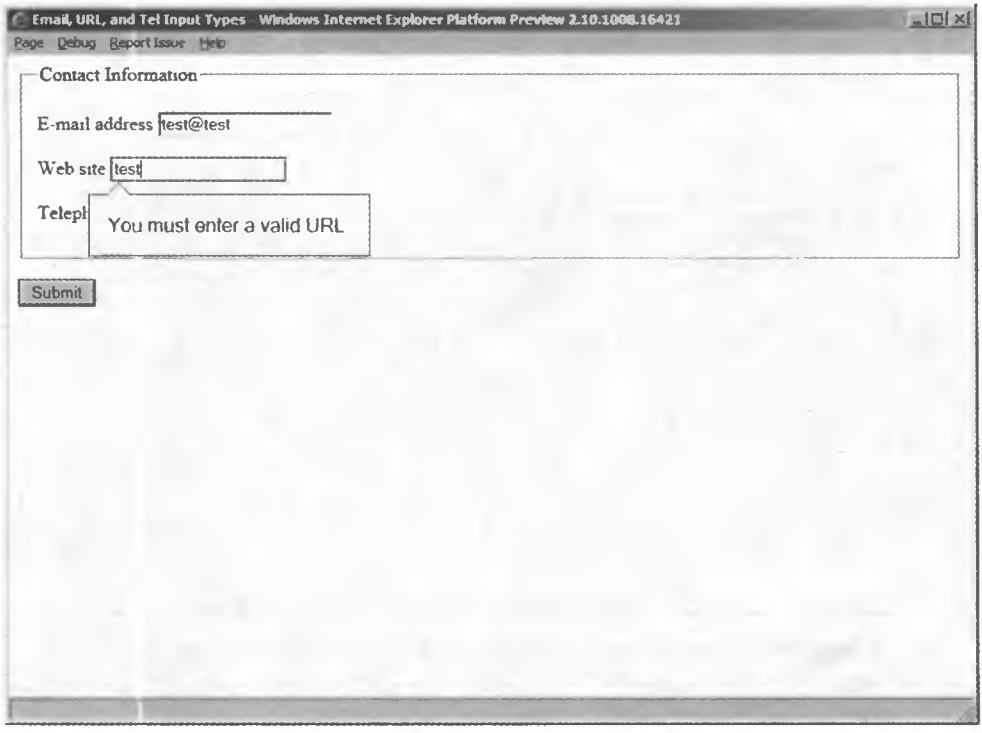

Рис. 3.6. Сообщение об ошибке в поле ввода типа url во втором Internet Explorer Platform Preview

По всему миру используются различные форматы телефонных номеров, поэтому поле ввода типа **tel** не имеет конкретного синтаксиса. Однако для определения строгого формата можно использовать новый атрибут **pattern**. Дополнительную информацию об этом можно получить в рецепте 3.12.

С точки зрения обратной связи, устанавливаемой с посетителем после заполнения формы, пользовательские агенты имеют несколько различий. На момент написания этой статьи отдельные браузеры уже начали визуально выделять поля с некорректными данными после того, как пользователь закончит ввод. Но они до сих пор не выводят сообщение об ошибке, пока форма не будет отправлена. Другие браузеры не обеспечивают обратную связь вплоть до отправки формы. Наличие поддержки этих возможностей браузерами описано в табл. 3.2.

Таблица 3.2. Браузеры, поддерживающие поля ввода типа email, url и tel

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	4.0+	10+	4.0+	11.0+	✓	✓

Настройка сообщений об ошибках по умолчанию. Если вы внимательно посмотрите на рис. 3.5 и 3.6, то заметите, что сообщение об ошибке по умолчанию зависит от браузера. Добавив атрибут **title** для каждого поля ввода, вы можете задать собственный вариант сообщений об ошибке:

```
<input type="url" name="website" title="Это не похоже на допустимый
    веб-адрес.">
```

Как и в случае с поиском, визуально трудно определить, действительно ли пользовательский агент поддерживает **email**, **url** или **tel**. Здесь также пригодится тест от Майка Тейлора (http://www.miketaylr.com/code/input-type-attr.html). Браузеры, которые не различают эти новые типы поля ввода, будут отображать их как текстовые поля.

На некоторых устройствах с сенсорным экраном пользовательский агент будет выводить для этих полей специальные области на клавиатуре. Например, при выборе поля типа **email** появятся не только буквы, но и символ **@**, а также различные варианты домена верхнего уровня, как показано на рис. 3.7. Здорово, да?

Рис. 3.7. На клавиатуре iPad доступны символ @ и варианты TLD

Дополнительная информация

Более подробные сведения о конфигурации экранной клавиатуры для этих полей можно найти в онлайн-книге Марка Пилгрима «Погружение в HTML5» в разделах http://diveintohtml5.info/forms.html#type-email и http://diveintohtml5.info/forms.html#type-url.

3.3. Использование полей ввода даты и времени

Проблема

Необходимо предоставить пользователю форму для ввода даты и/или времени, например, чтобы он мог назначить встречу.

Решение

В HTML5 предусмотрено несколько новых типов полей ввода для задания даты и времени.

Первый вариант — элемент **input** с атрибутом **type**, который имеет HTML5-значение **datetime**. Он отображает виджет-календарь для выбора даты и счетчик для времени (рис. 3.8):

```
<form>
    <p><label>Укажите дату и время <input type="datetime"
        name="dateAndTime"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Для того чтобы разделить ввод даты и времени, как показано на рис. 3.9 и 3.10, задайте отдельные поля **input** с атрибутами **type**, имеющими HTML5-значения **date** и **time**:

```
<form>
    <fieldset>
        <legend>Назначить встречу</legend>
        <p><label>Дата <input type="date" name="date"></label></p>
        <p><label>Время <input type="time" name="time"></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p>
</form>
```

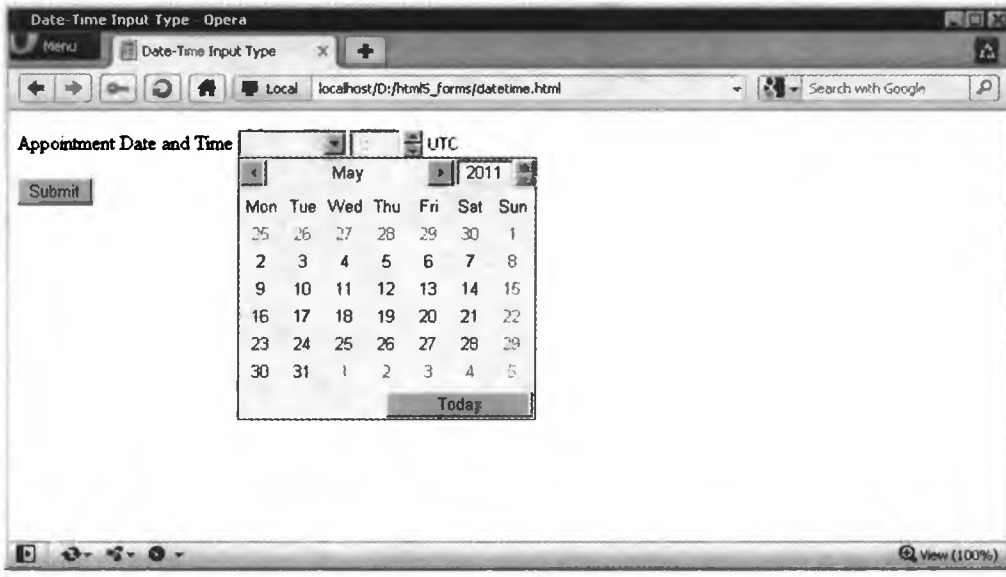

Рис. 3.8. Поле ввода datetime, отображающее виджет для выбора даты и времени в Opera 11.5

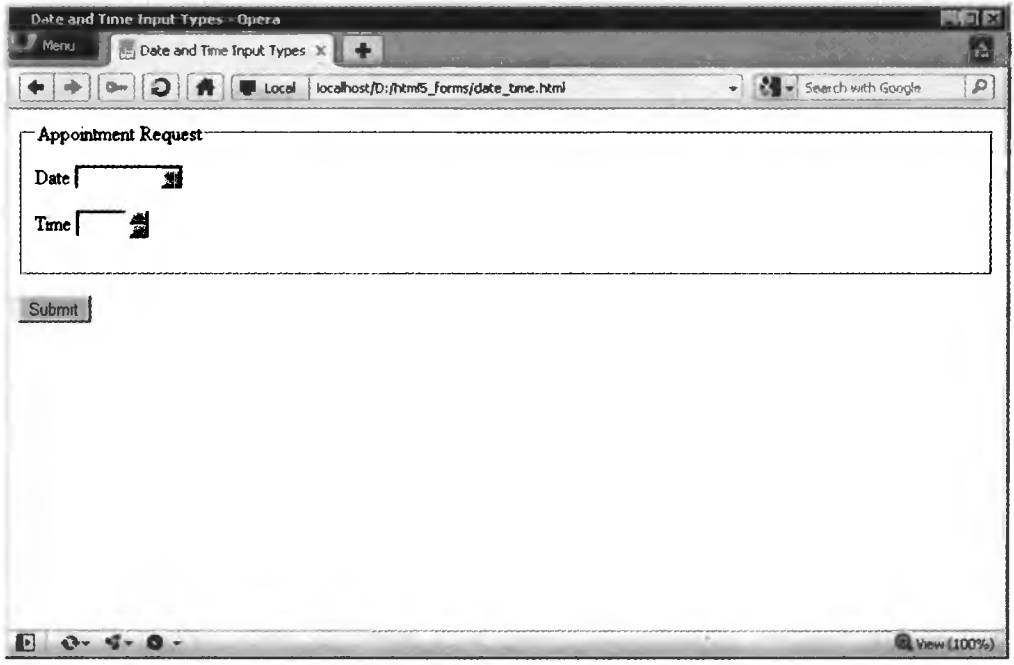

Рис. 3.9. Поля ввода date и time в Opera 11

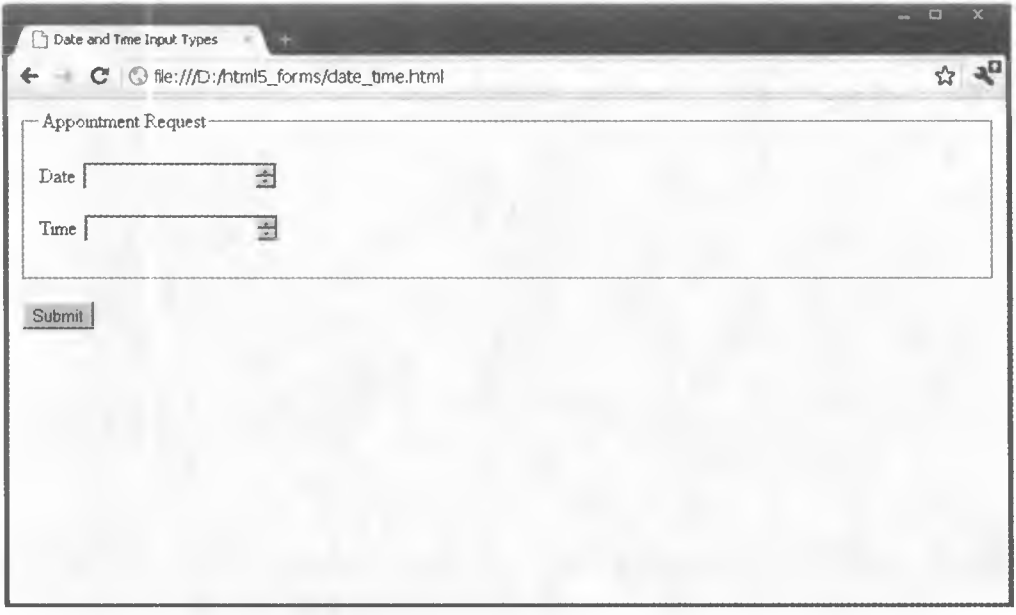

Рис. 3.10. Поля ввода date и time в Chrome 12

Обсуждение

Элементы **datetime**, **date** и **time** принадлежат к такой группе полей ввода, к которой также относятся типы **month**, **week** и **datetime-local**. Это в очередной раз подтверждает, что новые функции HTML5 основаны на многолетнем опыте по созданию библиотек JavaScript и других решений.

Браузер Opera всегда был лидером по созданию виджетов для управления датой и временем, представленных в виде календаря для поля типа **date** и счетчика для **time**. Многие ожидают, что другие браузеры в конечном счете последуют его примеру.

Недавно в настольных Webkit-браузерах появилась поддержка этих типов полей ввода со счетчиком. Пользовательские агенты, не поддерживающие их, по-прежнему отображают их как текстовые поля ввода.

Список браузеров, поддерживающих типы **date** и **time**, представлен в табл. 3.3.

Таблица 3.3. Поддержка браузерами полей ввода типов date и time

IE	Firefox	Chrome	Safari	Opera	iOS	Android
–	–	10+	5.0+	9.0+	✓	✓

Советы по обеспечению более высокого уровня кроссбраузерности можно найти в рецепте 3.14.

Как и для всех остальных полей ввода, вы можете указать для даты параметры ее отображения на странице. Значение типа **date** должно соответствовать определенному формату даты, например **2010-12-31**, то есть 31 декабря 2010 года.

Значение типа **time** должно отвечать определенному формату времени, например **23:59:59**, то есть 11:59 вечера, или за секунду до полуночи.

Наконец, значение типа **datetime** должно соответствовать формату, в котором объединены дата и время, например **2010-12-31T23:59:59Z**, то есть за секунду до полуночи по UTC 31 декабря 2010 года.

Расширенные возможности. Элемент **input** имеет два новых атрибута: **min** и **max**. Они позволяют ограничить диапазон даты и времени, которые могут быть выбраны пользователем (рис. 3.11).

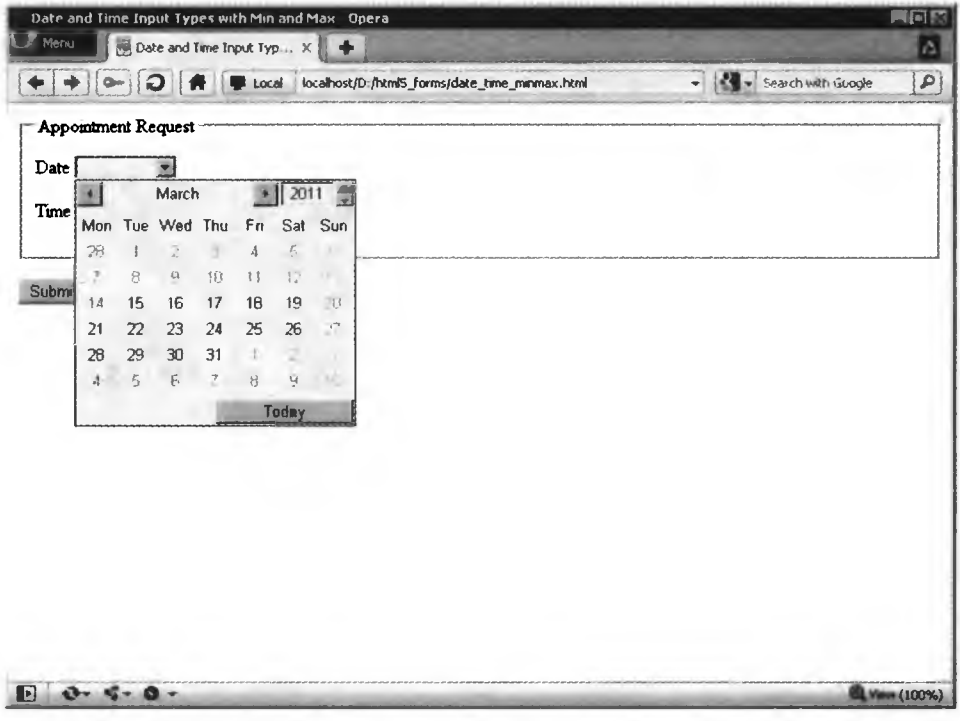

Рис. 3.11. Благодаря атрибуту min невозможно выбрать дату до 15 марта 2011 года (Opera 11.5)

Иначе говоря, пользователи не смогут назначить встречу в прошлом или указать дату рождения в будущем:

```
<form>
    <fieldset>
        <legend>Назначить встречу</legend>
        <p><label>Дата <input type="date" name="date" min="2011-03-15"
            max="2012-03-14"></label></p>
        <p><label>Время <input type="time" name="time" min="08:00"
            max="18:00"></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p>
</form>
```

Дополнительная информация

Для получения дополнительного материала о допустимых значениях даты и времени см. спецификацию HTML5: http://www.w3.org/TR/html5/common-microsyntaxes.html#dates-and-times.

3.4. Численный ввод

Проблема

Необходимо предоставить пользователю поле для ввода чисел, например, чтобы указать количество товаров в форме заказа.

Решение

Используйте элемент **input** с атрибутом **type**, имеющим HTML5-значение **number**:

```
<form>
    <p><label>Количество <input type="number"
        name="quantity"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Как вы уже догадались, значение поля ввода типа **number** должно быть целым числом, то есть нельзя указывать числа с плавающей точкой.

Если посетитель вводит недопустимое значение, то пользовательский агент предупредит его, и данные формы не будут отправлены (рис. 3.12).

Рис. 3.12. Числовой счетчик в Chrome 12

Некоторые пользовательские агенты отображают такое поле формы в виде счетчика. На отдельных устройствах с сенсорным экраном пользовательские агенты показывают для этого поля цифровую экранную клавиатуру.

Браузеры, поддерживающие поле ввода типа **number**, перечислены в табл. 3.4.

Таблица 3.4. Поддержка браузерами поля типа number

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	–	10+	4.0+	9.0+	✓	✓

Удобные поля ввода

Естественно, при реализации поля ввода типа **number** необходимо учитывать его назначение. Например, если вы предлагаете ввести почтовый индекс

США, то подобное поле в виде счетчика может сбить пользователя с толку, так как он не поймет причину такого выбора. С другой стороны, вы не можете гарантировать корректное отображение счетчика во всех браузерах, поэтому постарайтесь аккуратно указывать текст с пояснениями. Не используйте счетчик со стрелками, когда в нем нет необходимости!

Загрузите тестирование поддержки браузеров от Майка Тейлора (http://www.miketaylr.com/code/input-type-attr.html), чтобы определить, на каком уровне конкретный браузер поддерживает этот тип поля. Если пользовательский агент не сможет распознать поле типа **number**, то покажет вместо него текстовое поле ввода.

Расширенные возможности

Для поля типа **number** предусмотрены атрибуты **min** и **max**, ограничивающие вводимые значения. Кроме того, чтобы указать величину, на которую следует увеличить значение, можно определить атрибут **step**. К примеру, клиент должен заказать товар, поставляемый только в паре, поэтому минимальное количество равно 2, и шаг также имеет значение 2 (рис. 3.13).

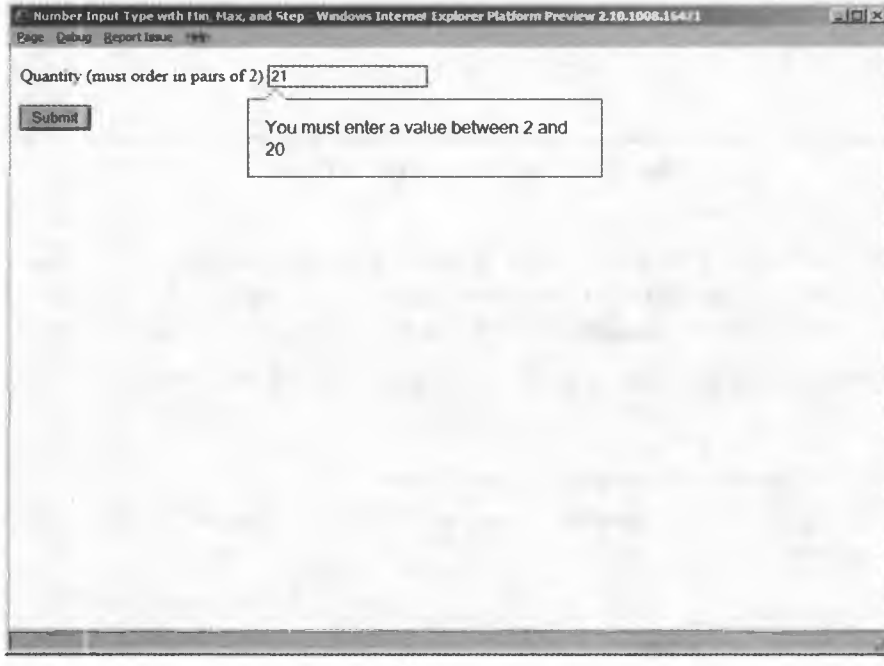

Рис. 3.13. Internet Explorer 10 Platform Preview 2 использует для типа number текстовое поле ввода, но поддерживает атрибуты min, max и step и, если необходимо, выдает сообщение об ошибке

```
<form>
    <p><label>Количество (необходимо заказывать 2 попарно) <input
        type="number" name="quantity" min="2" max="20"
        step="2"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

На рис. 3.14 представлен скриншот конфигурации экранной клавиатуры для поля типа **number**.

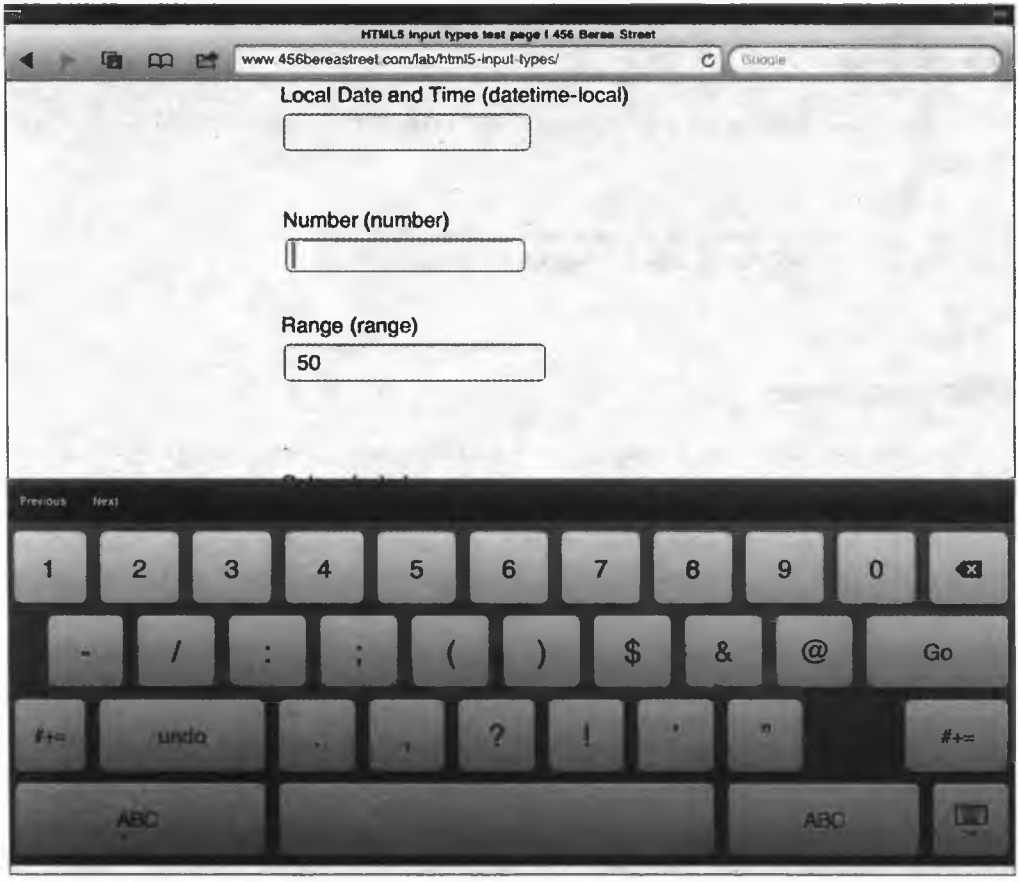

Рис. 3.14. Для поля ввода типа number отображается цифровая клавиатура

Дополнительная информация

Читайте статью Марка Пилгрима по поводу поля ввода типа **number**: http://diveintohtml5.info/forms.html#type-number.

3.5. Выбор значения из числового диапазона

Проблема

Необходимо предложить пользователю форму для выбора числа из диапазона, например, для регулировки громкости видео.

Решение

Используйте элемент **input** с атрибутом **type**, имеющим HTML5-значение **range**:

```
<form>
    <p><label>Громкость <input type="range" name="volume"
        min="0" max="10" step=".5" value="5"></label></p>
</form>
```

Обсуждение

Значение для поля типа **range**, как и для **number**, должно быть целым числом, однако точность здесь не важна.

Точность важна для форм заказа и т. п., для значений вроде количества, поэтому в таких случаях следует использовать **number**. Поле типа **range** может быть полезно для сравнительных или субъективных данных, таких как рейтинг события или продукта. Чтобы подчеркнуть это, спецификация требует устанавливать такое поле в виде слайдера или аналогичного элемента управления (рис. 3.15).

Webkit-браузеры и Opera отображают поле ввода типа **range** в виде горизонтального ползунка. Если пользовательский агент не поддерживает такой тип ввода, то он покажет текстовое поле.

Браузеры, поддерживающие поле ввода типа **range**, перечислены в табл. 3.5.

Таблица 3.5. Поддержка браузерами поля типа range

IE	Firefox	Chrome	Safari	Opera	iOS	Android
–	–	10+	4.0+	10.0+	✓	✓

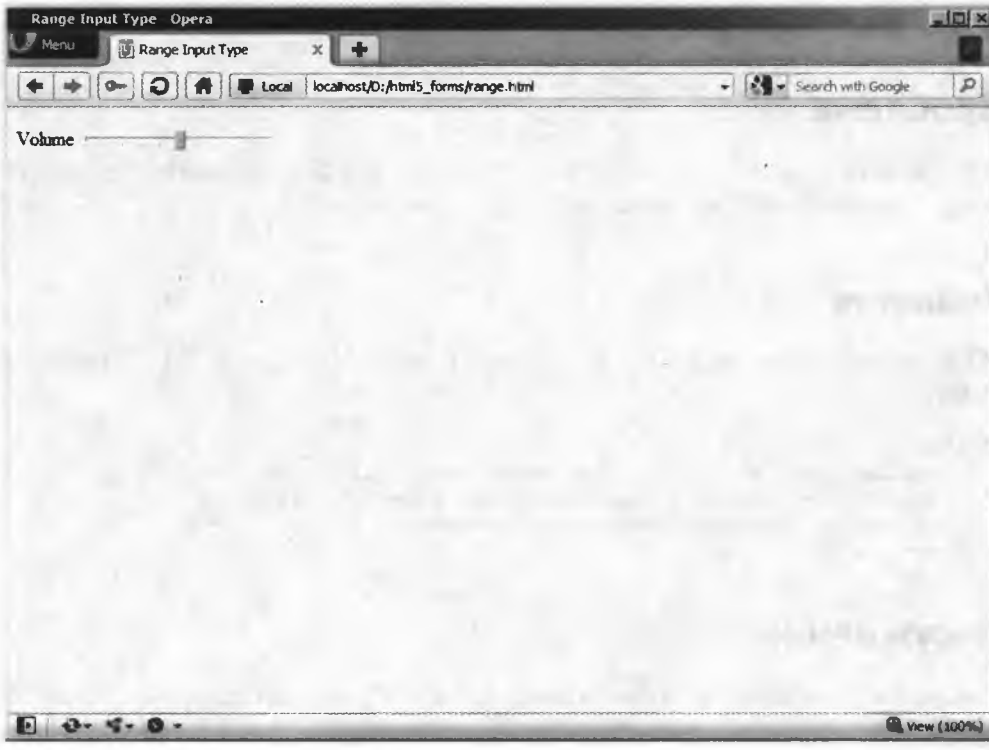

Рис. 3.15. Поле ввода типа range в Opera 11.5

Советы по обеспечению более высокого уровня кроссбраузерности можно найти в рецепте 3.14.

Расширенные возможности. Как и для **number**, для **range** можно указать минимальное и максимальное значение с помощью атрибутов **min** и **max**, а также величину приращения, используя атрибут **step**. Это позволит применять **range** вместо переключателей, широко употребляемых для определения рейтинга.

Дополнительная информация

W3C-спецификация **range**: http://www.w3.org/TR/html-markup/input.range.html.

3.6. Выбор цвета

Проблема

Необходимо предоставить пользователю форму для выбора цвета, например, для настройки внешнего вида веб-страницы.

Решение

Определите элемент **input** с атрибутом **type**, хранящим HTML5-значение **color**:

```
<form>
    <p><label>Цвет фона <input type="color" name="bg"></label></p>
    <p><label>Цвет шрифта <input type="color" name="fg"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Значение поля ввода **color** ограничивается шестнадцатеричным RGB-форматом с символом решетки, или октоторпом, **#**, указанным перед шестью цифрами.

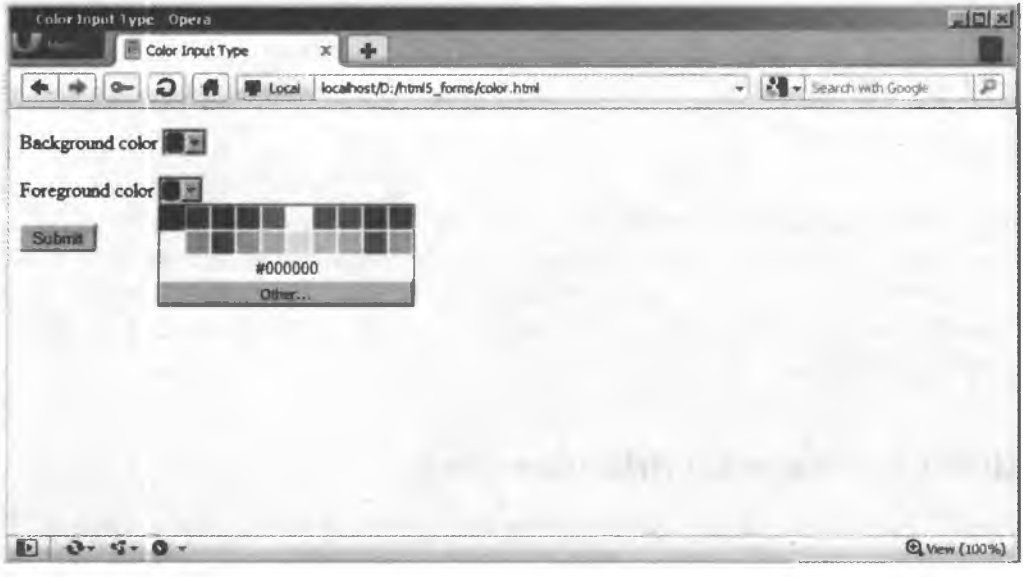

Рис. 3.16. Поле ввода типа color в Opera 11.5

Спецификация HTML5 предполагает использование для этого типа поля специальной палитры, но пока это возможно только в Opera. Webkit-браузеры, такие как Chrome и Safari, поддерживают **color** и показывают его в виде текстового поля ввода. Но при этом они успешно проверяют введенные данные и запрещают отправку формы, если указанное значение недопустимо (рис. 3.16 и 3.17).

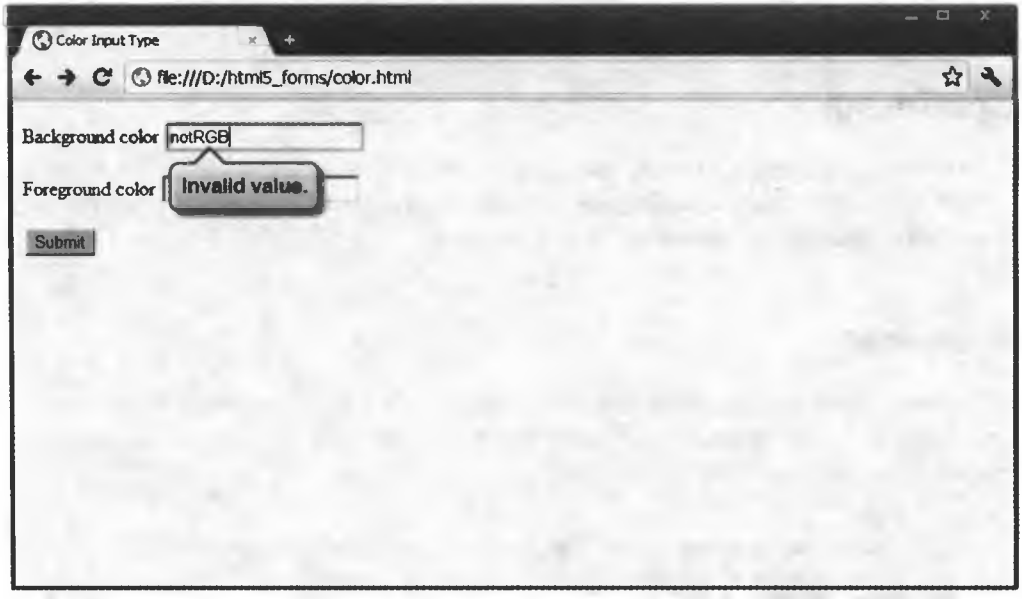

Рис. 3.17. Поле ввода типа color в Chrome 12

Если пользовательский агент не поддерживает поле ввода типа **color**, то отображается текстовое поле ввода.

Более подробная информация о поддержке браузерами поля типа **color** приведена в табл. 3.6.

Таблица 3.6. Поддержка браузерами поля ввода типа color

IE	Firefox	Chrome	Safari	Opera	iOS	Android
–	–	10+	5.0+	11.0+	✓	✓

Советы по обеспечению более высокого уровня кроссбраузерности можно найти в рецепте 3.14.

Дополнительная информация

«Октоторп» — хорошее слово, которое часто произносится в приличном обществе: http://en.wiktionary.org/wiki/octothorpe.

3.7. Создание редактируемого раскрывающегося списка

Проблема

Необходимо предоставить пользователю возможность вводить данные и предложить ему на выбор несколько вариантов. Иногда это называют *редактируемым раскрывающимся списком.*

Решение

Используйте для создания подобного списка HTML5-элемент **datalist** в совокупности с элементом **option**. Кроме того, свяжите список с элементом **input** через атрибут **list**:

```
<form>
    <p><label>Сумма пожертвования <input type="text" name="donation"
        list="donations"></label></p>
    <datalist id="donations">
        <option value="10.00">10.00</option>
        <option value="25.00">25.00</option>
        <option value="50.00">50.00</option>
    </datalist>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Элемент **datalist** применяется при создании списка возможных значений для других управляемых полей ввода — он не является формой ввода или управляющим элементом.

Возможные значения присваиваются с помощью элементов **option**, как это могло быть сделано для элемента **select**. Однако пока **datalist** не связан ни с каким элементом **input**, на экране ничего не происходит. Элемент **datalist** будет привязан к полю **input**, когда вы укажете для него идентификатор, используя в качестве значения атрибут **list** элемента **input**.

Подобный управляемый элемент обеспечивает ввод пользовательских данных и предоставляет выбор пункта из списка. Это мало чем отличается от опережающего ввода или функции автозаполнения, реализованной во многих браузерах и на большинстве поисковых сайтов (рис. 3.18 и 3.19).

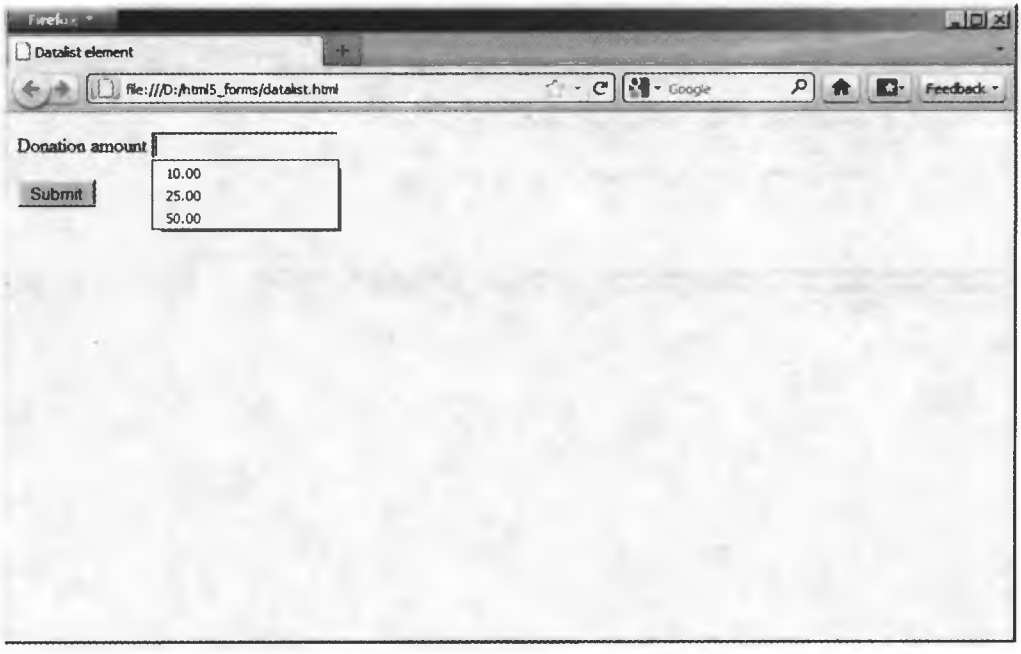

Рис. 3.18. Редактируемый список на основе input и datalist в Firefox 4

В тех браузерах, где до сих пор не реализован элемент **datalist**, результаты приведенного выше решения появятся на экране в текстовом поле ввода, а не в подобном списке. Устаревшие браузеры могут показывать метки с вариантами.

Обзор браузеров, поддерживающих **datalist**, приведен в табл. 3.7.

Таблица 3.7. Поддержка браузерами элемента datalist

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	4.0+	–	–	10.0+	–	–

В обход datalist. Веб-разработчик Джереми Кейт (Jeremy Keith) предложил вариант реализации списков, работающий как в современных браузерах, поддерживающих HTML5-элемент **datalist**, так и в тех, которые этого не делают:

```
<form>
    <p><label for="donation">Сумма пожертвования </label>
    <datalist id="donations">
        <select name="donation">
            <option></option>
            <option value="10.00">10.00</option>
            <option value="25.00">25.00</option>
            <option value="50.00">50.00</option>
            <option>Другое</option>
        </select>
        Если выбрано "Другое", пожалуйста, укажите свой вариант:
    </datalist>
    <input type="text" id="donation" name="donation"
        list="donations"></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Рис. 3.19. Редактируемый список на основе input и datalist в Internet Explorer 10 Platform Preview 2

Полученный результат валиден, семантически полон и элегантен: в первом случае отображается редактируемый раскрывающийся список, а в последнем — рядом со списком оказывается поле ввода, как показано на рис. 3.20.

Рис. 3.20. Запасное решение для элемента datalist от Джереми Кейта в Chrome 12

Дополнительная информация

Полный текст статьи, посвященной решению для **datalist** от Джереми Кейта, вы найдете по адресу http://adactio.com/journal/4272/. Для получения информации о поддержке **datalist** см. http://caniuse.com/datalist.

3.8. Обязательные поля формы

Проблема

Необходимо обеспечить заполнение обязательных полей формы перед ее отправкой.

Решение

Используйте HTML5-атрибут **required** для всех полей формы, которые необходимо заполнить, чтобы отправить форму:

```
<form>
    <fieldset>
        <legend>Авторизация</legend>
        <p><label>Имя пользователя <input type="text" name="username"
            required></label></p>
        <p><label>Пароль <input type="password" name="pwd"
            required></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Атрибут **required** можно добавить для нескольких элементов в форме или не указывать вовсе.

Когда пользователь отправляет форму, браузер проверяет, не оставлены ли пустыми все обязательные поля. При необходимости он остановит отправку данных и выведет сообщение об ошибке. Внешний вид сообщения об ошибке варьируется в зависимости от браузера (рис. 3.21–3.24).

Рис. 3.21. Атрибут required сообщает об ошибке в Firefox 4

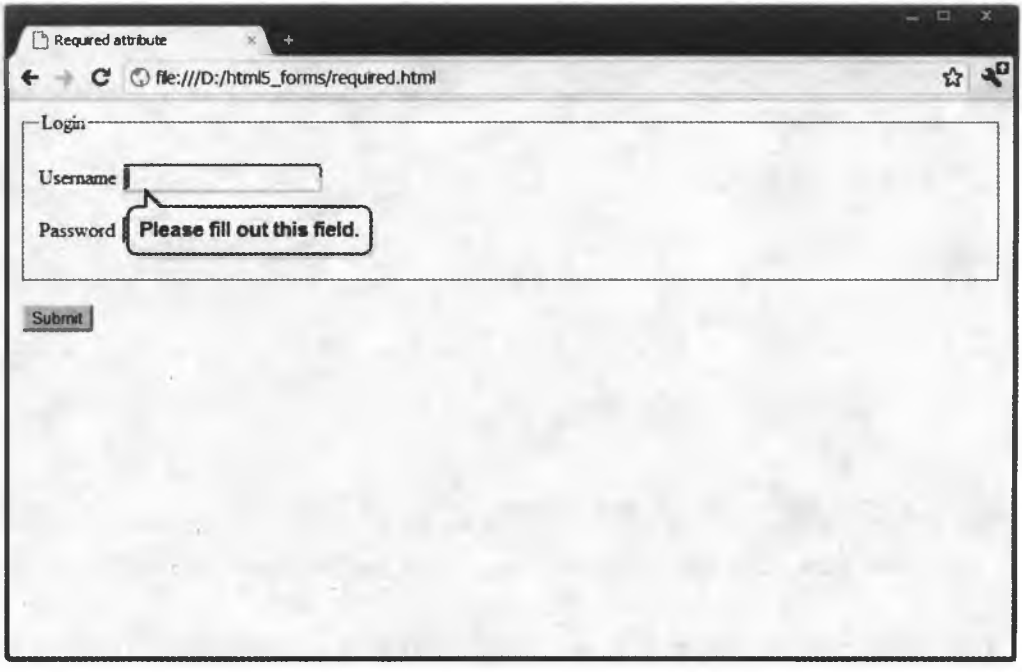

Рис. 3.22. Атрибут required сообщает об ошибке в Chrome 12

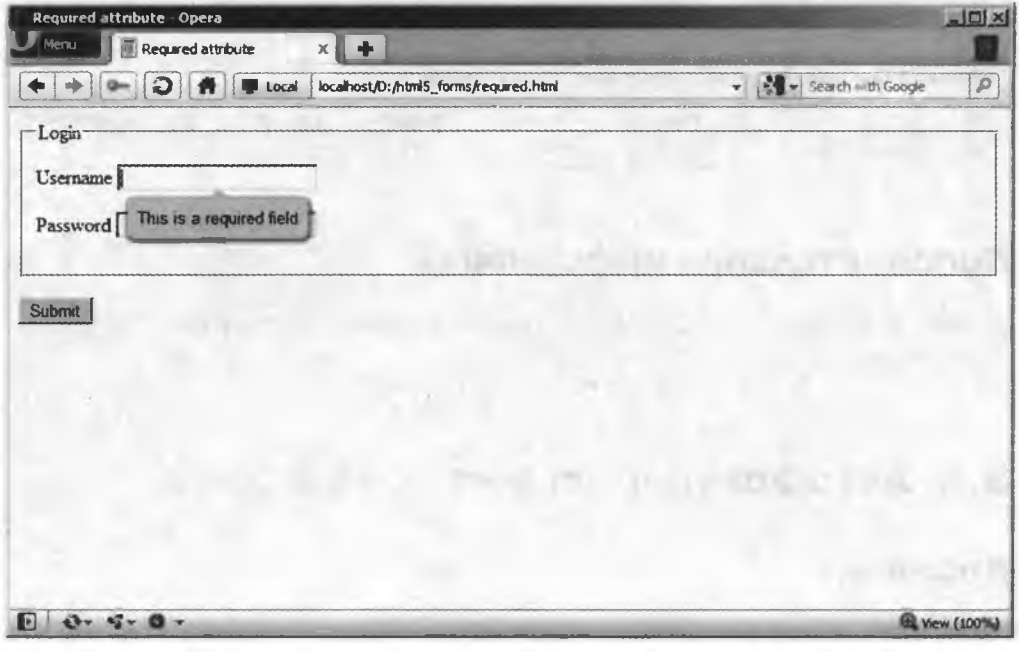

Рис. 3.23. Атрибут required сообщает об ошибке в Opera 11.5

Рис. 3.24. Атрибут required сообщает об ошибке в IE 10 Platform Preview 2

Если атрибут **required** не поддерживается пользовательским агентом, то он будет проигнорирован. Однако для проверки с помощью этого атрибута вы еще можете создать сценарий JavaScript (подробнее см. рецепт 3.14).

Список браузеров, поддерживающих **required**, приведен в табл. 3.8.

Таблица 3.8. Браузеры, поддерживающие атрибут required

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	4.0+	10+	5.0+	10.0+	✓	✓

Дополнительная информация

Узнайте, поддерживает ли браузер проверку полей формы: http://caniuse.com/form-validation.

3.9. Автофокусировка на поле формы

Проблема

При загрузке страницы необходимо акцентировать внимание на той или иной области формы.

Решение

Укажите HTML5-атрибут **autofocus** только для одного поля формы:

```
<form>
    <p><label>Найти <input type="search" name="query"
        autofocus></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Одни разработчики любят автофокус, другие его ненавидят. Очень хорошо, что Google автофокусируется на поле поиска, но это откровенно раздражает, когда нужно нажатием кнопки прокрутить страницу вниз.

Существует простое и доступное решение, которое избавит вас от необходимости писать сценарий JavaScript. Помните, что атрибут **autofocus** может быть указан только один раз в *документе*, а не в форме! Для обеспечения удобства работы в пользовательском агенте можно задать настройки или изменить фокусировку.

Если атрибут **autofocus** не поддерживается пользовательским агентом, то он будет проигнорирован. Для автофокусировки в определенном поле вы можете использовать JavaScript.

Если хотите ознакомиться с поддержкой браузерами атрибута **autofocus**, обратите внимание на табл. 3.9.

Таблица 3.9. Поддержка браузерами атрибута autofocus

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	4.0+	3.0+	4.0+	10.0+	✓	✓

Дополнительная информация

Подробности см. в рецепте 3.14.

3.10. Отображение замещающего текста

Проблема

Необходимо показать в текстовом поле совет или подсказку.

Решение

Для отображения пояснительного текста, когда пользовательский фокус не находится в поле формы, добавьте HTML5-атрибут **placeholder**:

```
<form>
    <fieldset>
        <legend>Контактная инфомация</legend>
        <p><label>Адрес электронной почты <input type="email"
            name="email" placeholder="user@domain.com"></label></p>
        <p><label>Сайт <input type="url" name="website"
            placeholder="http://www.domain.com/"></label></p>
        <p><label>Телефон <input type="tel" name="phone"
            placeholder="123-123-1234"></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

Специалисты и разработчики пользовательских интерфейсов, стремящиеся к практичности своих проектов, в течение длительного времени пытались решить проблему отображения в форме ненавязчивого текста подсказок.

За последние пять лет замещающий текст в поле ввода стал стандартным дизайнерским решением, и теперь у нас появился легкий способ его реализации. Если в элементе **input** задан атрибут **placeholder**, то пользовательский агент будет отображать в поле формы значение этого атрибута.

Замещающий текст может быть включен или выключен в зависимости от того, на каком элементе страницы находится фокус и есть ли данные в поле формы: когда пользователь сфокусируется на поле, замещающий текст исчезнет; если же пользователь покинет пустое поле формы, то появится замещающий текст (рис. 3.25).

Атрибут **placeholder** делает работу с формой удобнее. Его основная цель — предоставить дополнительные подсказки для пользователя. Для управления подписями на форме не стоит пренебрегать элементом **label**.

Если пользовательский агент не поддерживает **placeholder**, то атрибут игнорируется.

Информация о поддержке атрибута **placeholder** представлена в табл. 3.10.

Таблица 3.10. Поддержка браузерами атрибута placeholder

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	4.0+	4.0+	4.0+	11.0+	✓	✓

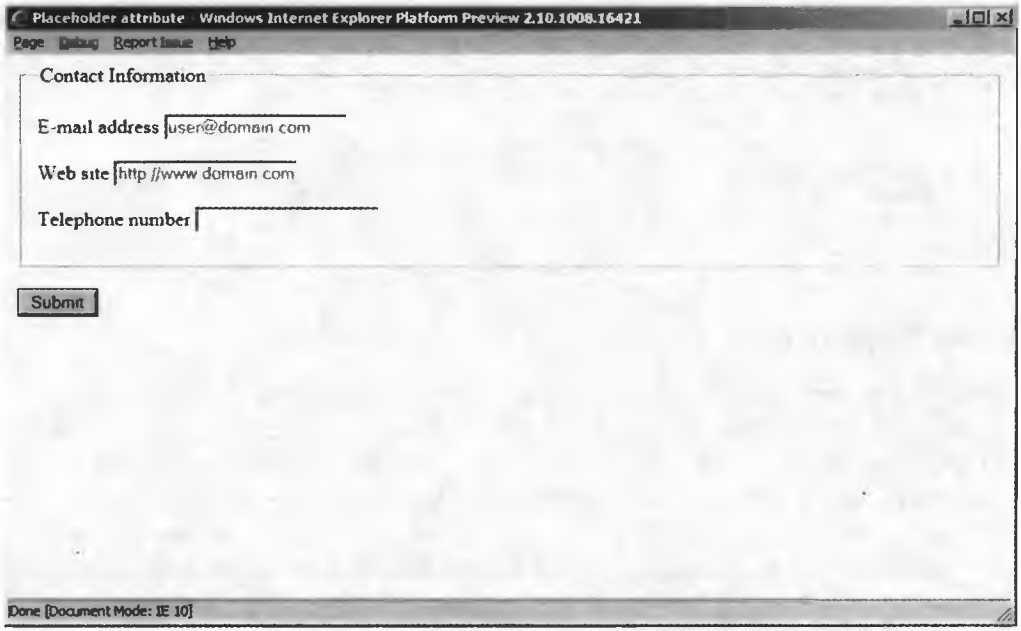

Рис. 3.25. Замещающий текст в Internet Explorer 10 Platform Preview 2

Дополнительная информация

Если хотите обеспечить более согласованное использование этого атрибута в браузерах, которые его поддерживают, и тех, которые этого не делают, читайте рецепт 3.13.

3.11. Отключение автозаполнения

Проблема

Необходимо отключить возможность автоматического заполнения полей формы.

Решение

Присвойте HTML5-атрибуту **autocomplete** значение **"off"** для любого индивидуализированного поля, например поля для ввода пароля:

```
<form>
    <fieldset>
        <legend>Авторизация</legend>
        <p><label>Имя пользователя <input type="text"
            name="username"></label></p>
        <p><label>Пароль <input type="password" name="pwd"
            autocomplete="off"></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p> +
</form>
```

Обсуждение

Атрибут **autocomplete** был введен в Internet Explorer в 1999 году и позже принят в других браузерах, хотя не был частью какой-либо спецификации HTML или XHTML. По умолчанию для всех полей формы автозаполнение включено (**on**). В этом случае вы можете положиться на свой браузер, который будет помнить ваши пароли и благодаря автозаполнению поможет быстро зайти на посещаемые вами сайты. Однако если указать для этого атрибута значение **off**, то такой возможности не будет.

Вы можете отключить **autocomplete** как на уровне отдельных полей ввода, так и на уровне формы. Во втором случае допускается включить его для отдельного поля, установив **autocomplete="on"**.

Хотя многие эксперты по безопасности предлагают применять **autocomplete="on"** для полей с конфиденциальными данными, вы должны иметь в виду, что такие меры безопасности *не* очень эффективны. Некоторые браузеры пока не поддерживают **autocomplete**. Кроме того, существует множество способов обойти **autocomplete="off"** — есть инструменты, по-прежнему хранящие пароль для автоматического ввода.

Браузеры, не поддерживающие **autocomplete**, просто полностью игнорируют этот атрибут.

Список браузеров, поддерживающих **autocomplete**, приведен в табл. 3.11.

Таблица 3.11. Поддержка браузерами атрибута autocomplete

IE	Firefox	Chrome	Safari	Opera	iOS	Android
Да*	4+	Да*	Да*	10.0+	Да*	Да*

В табл. 3.11 «Да» означает, что нестандартные пути autocomplete были реализованы в браузере до появления HTML5. Откройте страницу Майка Тейлора http://www.miketaylr.com/code/input-type-attr.html, чтобы определить, когда браузеры стали поддерживать этот атрибут в стандарте HTML5.

Дополнительная информация

Атрибут **autocomplete** в спецификации WHATWG HTML: http://www.whatwg.org/specs/web-apps/current-work/multipage/forms.html#attr-form-autocomplete.

3.12. Ограничение значений

Проблема

Необходимо ограничить значения поля ввода в соответствии с указанным набором правил.

Решение

Добавьте HTML5-атрибут **pattern**, чтобы указать регулярное выражение для проверки пользовательского ввода:

```
<form>
    <p><label>Телефон <input type="tel" name="phone"
        pattern="[2-9][0-9]{2}-[0-9]{3}-[0-9]{4}"
        title="Северо-американский формат: XXX-XXX-XXXX"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Обсуждение

В течение многих лет веб-специалистам приходилось создавать сценарии, проверяющие пользовательские данные до того, как форма будет отправлена, и большинство из нас, вероятно, скажет, что ненавидит это.

В HTML5 для проверки формы ввода существует метод, не основаный на JavaScript, — *регулярные выражения*.

Регулярные выражения покажутся вам очень мощным и интересным инструментом, как только вы научитесь их создавать. Нет необходимости корпеть над огромной книгой вроде «Регулярные выражения» Джеффри Фридла от O'Reilly. Издание «Регулярные выражения: список рецептов» (также от O'Reilly) легко поможет вам понять, как описать общие выражения. Вы получите тот же эффект, что и от прочтения огромной книги, но в кратчайшие сроки.

HTML5-атрибут **pattern** позволяет определять регулярное выражение для поля ввода. После завершения ввода в поле или перед отправкой формы

(реализация зависит от браузера) указанные пользователем данные будут сравниваться с шаблоном, и если они не совпадут, появится сообщение об ошибке и форма не будет отправлена (рис. 3.26).

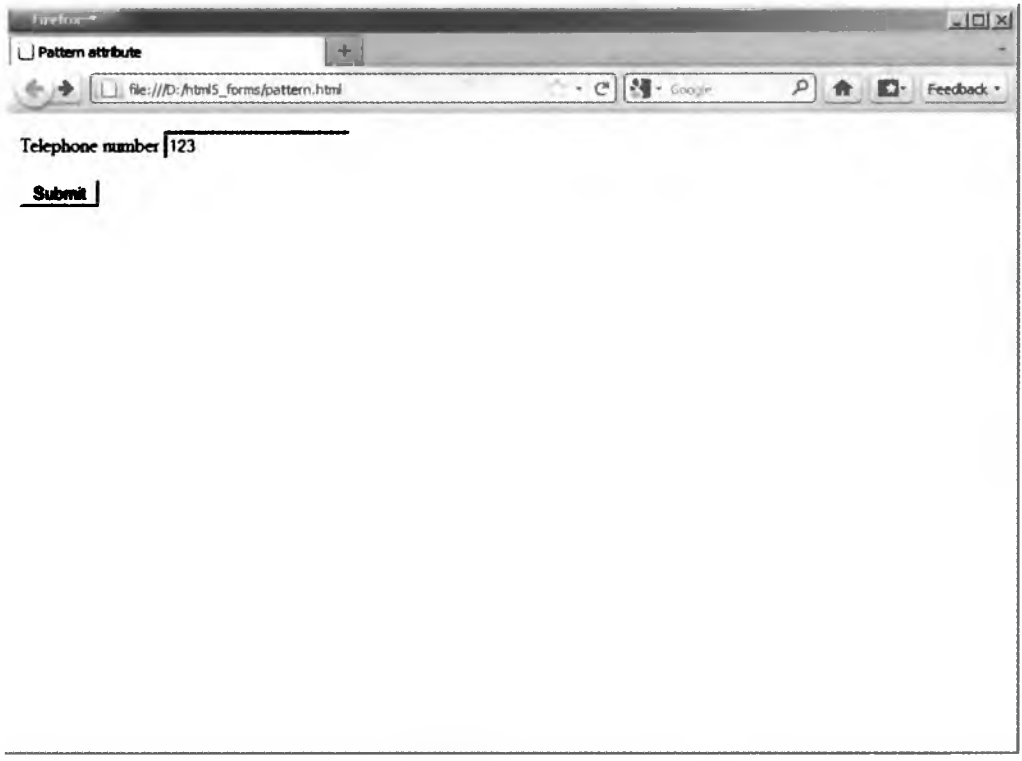

Рис. 3.26. Атрибут pattern выделяет поле ввода с неточными данными в Firefox 4, а сообщение об ошибке появится перед отправкой формы

В регулярных выражениях **pattern** предусмотрен тот же синтаксис, что и в JavaScript. Имейте в виду, что атрибут **pattern** должен хранить полное значение, иначе пользователь увидит сообщение об ошибке.

Применяя атрибут **pattern**, вы должны также указывать атрибут **title** для описания шаблона. Пользовательский агент может отображать этот текст как подсказку или сообщение об ошибке, если введеное значение не соответствует шаблону (рис. 3.27).

Если пользовательский агент не может проанализировать регулярное выражение, то оно будет проигнорировано и поле останется непроверенным. Если пользовательский агент не поддерживает атрибут **pattern**, то проверки также не будет.

Рис. 3.27. Атрибут title выводит подсказку в Internet Explorer 10 Platform Preview 2

Сведения о поддержке браузерами атрибута **pattern** приведены в табл. 3.12.

Таблица 3.12. Поддержка браузерами атрибута pattern

IE	Firefox	Chrome	Safari	Opera	iOS	Android
10 Platform Preview 2	4+	10+	5+	11+	–	–

Советы по обеспечению более высокого уровня кроссбраузерности можно найти в рецепте 3.14.

Дополнительная информация

Как и в случае с **required**, проверьте возможность работы в браузерах атрибута **pattern**: http://caniuse.com/form-validation. Список часто используемых регулярных выражений приведен по адресу http://html5pattern.com.

3.13. Поддержка работы HTML5 в устаревших браузерах

Проблема

Необходимо, чтобы HTML5-атрибуты и типы полей ввода работали в браузерах, не поддерживающих HTML5.

Решение

Используйте JavaScript-библиотеку Modernizr (http://www.modernizr.com), чтобы определить наличие поддержки HTML5-атрибутов, а затем разработать или реализовать альтернативное решение, например JQuery UI (http://jqueryui.com), если поддержка отсутствует.

Modernizr — небольшая библиотека JavaScript с открытым исходным кодом, которая определяет наличие реализации HTML5 и CSS3 в браузерах. Вместо того чтобы проконтролировать наличие поддержки браузером в целом, Modernizr выполняет проверку для каждой функции в отдельности (см. рецепт 2.5). Modernizr предоставляет очень надежный способ обнаружить поддержку возможностей HTML5-форм.

JQuery UI — это библиотека, реализующая шаблон проектирования пользовательского интерфейса. Она основана на библиотеке JavaScript JQuery, которая используется многими самыми популярными сайтами в Интернете, в том числе Google, Amazon, Twitter и Microsoft. JQuery облегчает манипулирование DOM, а JQuery UI позволяет легко реализовать применение виджетов.

Пример 1: поддержка атрибута autofocus

Сохраните Modernizr в свою локальную папку и объявите в элементе **head** своего документа:

```
<head>
    <script src="modernizer.js"></script>
</head>
```

Код формы, использующей атрибут **autofocus**:

```
<form>
    <p><label>Search <input type="search" name="query" id="query"
        autofocus></label></p>
    <p><button type="submit">Submit</button></p>
</form>
```

Создайте сценарий, определяющий наличие поддержки атрибута **autofocus** и, если ее нет, использующий JavaScript, чтобы сфокусироваться на элементе. Этот сценарий можно добавить сразу после формы, чтобы как можно быстрее вызвать событие фокусировки:

```
<script>
    if (!Modernizr.input.autofocus) {
        document.getElementById("query").focus();
    }
</script>
```

Пример 2: поддержка атрибута placeholder

Объявите Modernizr и JQuery в элементе **head** своего документа:

```
<head>
    <script src="modernizer.js"></script>
    <script src="jquery.js"></script>
</head>
```

Код вашей формы с атрибутом **placeholder**:

```
<form id="search">
    <p><label>Найти <input type="search" name="query" id="query"
        value="" placeholder="Введите запрос"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Создайте сценарий, определяющий поддержку атрибута **placeholder** и, если ее нет, использующий JavaScript для отображения замещающего текста в поле ввода:

```
<script>
    if (!Modernizr.input.placeholder) {
        $("#query").focus(function() {
            if ($("#query").val() == $("#query").attr('placeholder')) {
                $("#query").val("");
            }
        });
        $("#query").blur(function() {
            if ($("#query").val() == "") {
                $("#query").val($("#query").attr('placeholder'));
            }
        });
        $("#query").blur();
        $("#search").submit(function() {
            if ($("#query").val() ==
        $("#query").attr('placeholder')) {
            return false;
            }
        });
    }
</script>
```

Даже если вы никогда прежде не работали с JQuery, то, вероятно, очень легко прочитаете приведенный выше код. JQuery использует синтаксис CSS-селекторов для обозначения DOM-элементов. Так, **$("#query")** обращается к поисковому полю ввода через **id**. Методы **focus()** и **blur()** связываются с полем поиска, создавая обработчики событий, когда поле оказывается в фокусе или теряет его. Наконец, при отправке поисковой формы проверяется значение, введенное в поле поиска: нужно выяснить, не является ли оно копией замещающего текста. Если это так, то форма не отправляется.

Когда срабатывает событие **blur**, полю поиска (**val**) присваивается значение в виде содержимого атрибута **placeholder** при условии, что изначально значение было пустым.

Когда срабатывает событие **focus**, в качестве значения поля поиска устанавливается пустая строка, но только при условии, что сначала значение было таким же, как и текст атрибута **placeholder**.

Этот сценарий можно вставить сразу после формы или объединить с другими сценариями внизу элемента **body** документа.

Эти методы прекрасно работают со всеми типами полей, кроме поля для ввода пароля. Браузеры скрывают значение такого поля, поэтому и замещающий текст оказывается спрятан. Если вам необходимо показать замещающий текст в поле ввода пароля, используйте JavaScript. Это позволит указать текст не как значение поля ввода.

Пример 3: поддержка поля ввода типа date

Объявите Modernizr, JQuery и JQuery UI в разделе **head** своего документа (не забудьте CSS для JQuery UI!):

```
<head>
    <script src="modernizr.js"></script>
    <script src="jquery.js"></script>
    <script src="jquery-ui.js"></script>
    <link href="jquery-ui.css" rel="stylesheet">
</head>
```

Код формы с полем ввода типа **date**:

```
<form>
    <p><label>Дата рождения <input type="date" name="dob"
        id="dob"></label></p>
    <p><button type="submit">Отправить</button></p>
</form>
```

Создайте сценарий, определяющий наличие поддержки атрибута **date** и, если ее нет, использующий jQuery UI для запуска виджета календаря:

```
<script>
    if (!Modernizr.inputtypes.datetime) {
        $("#dob").datepicker();
    }
</script>
```

Бьюсь об заклад, вы удивлены, как мало кода потребовалось, например, по сравнению с предыдущим решением! В этом и состоит прелесть универсальных решений, которые предоставляет JQuery UI. На рис. 3.28 показаны результаты для браузера, не поддерживающего поле ввода типа **date**.

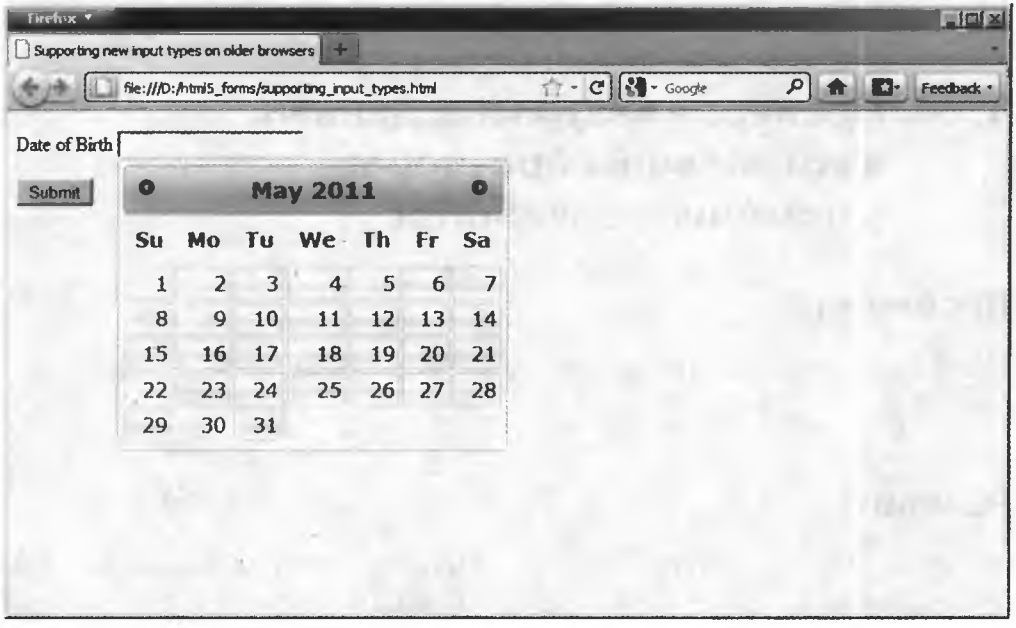

Рис. 3.28. Виджет выбора даты от JQuery UI включается в браузер Firefox 4, не поддерживающий поля ввода типа date и time

Как и в предыдущем примере, этот сценарий можно добавить сразу после формы или объединить с другими сценариями внизу раздела **body**.

Обсуждение

Помните, что новые HTML5-типы полей ввода основаны на решениях, неоднократно опробованных разработчиками HTML и JavaScript. Значит,

довольно легко можно найти UI-библиотеки, обеспечивающие замену функциональности для браузеров, которые пока не поддерживают HTML5-формы.

В будущем, когда все пользователи перейдут на браузеры, обеспечивающие работу HTML5-элементов, типов полей ввода и атрибутов, вы сможете быстро и легко избавиться от сценариев, не меняя разметку.

Дополнительная информация

Полный список альтернативных решений: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills.

3.14. Проверка вводимых данных в устаревших браузерах с помощью JavaScript

Проблема

Необходимо проверять вводимые данные в браузерах, не поддерживающих HTML5.

Решение

Используйте JavaScript-библиотеку Modernizr (http://www.modernizr.com), определяющую наличие поддержки HTML5-атрибутов и содержащую сценарии альтернативных решений для случаев, когда соответствующие функции не поддерживаются. Для получения информации о Modernizr и других библиотеках JavaScript вернитесь к рецепту 3.13.

Сохраните Modernizr в свою локальную папку и объявите в разделе **head** HTML-документа. Объявите также библиотеку JQuery (http://www.jquery.com), которая облегчит обращение к элементам формы и поможет присоединить обработчик событий:

```
<head>
    <script src="modernizer.js"></script>
    <script src="jquery.js"></script>
</head>
```

Код формы с HTML5-атрибутами **required** и **pattern** для проверки вводимых данных:

```
<form>
    <fieldset>
        <legend>Авторизация</legend>
        <p><label>Имя пользователя <input type="text" name="username"
            required></label></p>
        <p><label>Пароль <input type="password" name="password"
            required pattern="[0-9a-zA-Z]{12}" title="Должен состоять из
            12 буквенно-цифровых символов"></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p>
</form>
```

Создайте сценарий, определяющий наличие поддержки **required** и **pattern** и задающий альтернативные обработчики формы, если поддержка недоступна:

```
<script>
if (!Modernizr.input.required || !Modernizr.input.pattern) {
    $('form').submit(function() {
        var validData = true;
        $('[required], [pattern]').each(function() {
            if (($(this).attr('required') !== false) &&
                    ($(this).val() == "")){
                $(this).focus();
                alert($(this).attr('name') + " – обязательное поле!");
                validData = false;
                return false;
            }
            if ($(this).attr('pattern')){
                var regexp = new RegExp($(this).attr('pattern'));
                if (!regexp.test($(this).val())){
                    $(this).focus();
                    alert("Данные из поля " + $(this).attr('name') +
                            " имеют неправильный формат!");
                    validData = false; return false;
                }
            }
        });
        return validData;
    });
}
</script>
```

Обсуждение

Мы около 15 лет проверяли формы с помощью JavaScript, поэтому существует множество решений, обеспечивающих обязательное заполнение полей и проверку вводимых данных. Новый способ — использование

HTML5-атрибутов **required** и **pattern**, устанавливающих правила проверки. В недавнем прошлом разработчики применяли для подобных целей атрибут **rel**, но он больше не нужен (и недействителен в HTML5).

Опять же, когда все пользователи перейдут на браузеры, поддерживающие HTML5-формы, вы сможете легко и быстро избавиться от сценариев без необходимости изменять разметку.

Дополнительная информация

Smashing Magazine собрал лучшие примеры и обучающие программы для проверки веб-форм: http://www.smashingmagazine.com/2009/07/07/web-form-validation-best-practices-and-tutorials/.

3.15. Пример простой формы

Проблема

Необходимо предоставить пользователю форму для планирования событий, например, чтобы записаться на прием к врачу.

Решение

```
<!DOCTYPE html>
<html>
<head>
<title>Расписание приема</title>
</head>
<body>
<h1>Расписание приема</h1>
<form>
    <fieldset>
        <legend>Личная информация</legend>
        <p><label>Имя <input type="text" name="name"
            required></label></p>
        <p><label>Телефон <input type="tel" name="phone" required
            pattern="[2-9][0-9]{2}-[0-9]{3}-[0-9]{4}" title=
            "Северо-американский формат: XXX-XXX-XXXX"></label></p>
        <p><label>Адрес электронной почты <input type="email"
```

```
            name="email"></label></p>
        <p><label>Дата рождения <input type="date" name="dob">
            </label></p>
    </fieldset>
    <fieldset>
        <legend>Назначить прием</legend>
        <p><label for="reason">Какова причина вашего визита? <label>
        <datalist id="reasons">
            <select name="reason">
                <option>Плановый осмотр</option>
                <option>Симптомы простуды или гриппа</option>
                <option>Анализ крови</option>
                <option>Послеоперационный осмотр</option>
                <option>Другое</option>
            </select>
        Если выбрано "Другое", пожалуйста, укажите свой вариант:
        </datalist>
        <input id="reason" name="reason" list="reasons"></p>
        <p><label>Степень болевых ощущений <input type="range"
            name="pain" min="0" max="10" step=".5" value="5" title="0
            нет боли, 10 – боль нестерпима"></label></p>
        <p><label> Предпочтительная дата <input type="date" name="date"
            required min="2011-03-15" max="2012-03-14"></label></p>
        <p><label> Предпочтительное время <input type="time" name="time"
            required min="08:00" max="18:00" step="1:00"></label></p>
    </fieldset>
    <p><button type="submit">Отправить</button></p>
</form>
</body>
</html>
```

Обсуждение

Друзья, это будущее разметки форм! При просмотре в браузере, полностью поддерживающем HTML5-формы, этой странице потребуется лишь немного CSS — JavaScript больше не нужен! Формы будут красиво отображаться на самых разных устройствах и станут гораздо доступнее, чем сейчас (рис. 3.29).

Когда же будет обеспечена полная поддержка HTML5-форм в большинстве браузеров? Opera считается лидером в поддержке новых возможностей, но Firefox и Chrome постоянно выпускают новые версии и быстро догоняют этот браузер.

Второй Internet Explorer Platform Preview был выпущен во время написания этой статьи. Он также показывает хорошие результаты.

Рис. 3.29. Простая форма с основными стилями в Opera 11.5

Дополнительная информация

Стили формы подробно описаны в главе 8 книги Кристофера Шмитта «CSS: Рецепты программирования» (O'Reilly).

4 Встроенное аудио

Эмили Льюис

4.0. Введение

Почти каждый день мы имеем дело со *встроенным контентом* — информацией, которая импортируется или вставляется на веб-страницу (http://www.w3.org/TR/html5/content-models.html#embedded-content-0). Вспомните хотя бы элемент **img**, который с помощью атрибута **src** добавляет на веб-страницу изображения.

В HTML5 появилось гораздо больше вариантов встроенного контента, в том числе аудио, добавляемого с помощью нового элемента **audio** (http://dev.w3.org/html5/markup/audio.html).

Встроенные объекты? Да. Больше не будет неудобных элементов **object** и **embed**. Не придется вставлять аудио с помощью сторонних плагинов и не будет головной боли, связанной с разработкой динамических макетов или раскрывающихся меню.

Имея в арсенале элемент **audio**, вы сможете не только выкладывать аудиофайлы непосредственно в браузере, но и управлять стилем и атрибутами этого элемента с помощью CSS и JavaScript.

4.1. Добавление HTML5-аудио

Проблема

Необходимо воспроизводить на веб-странице встроенное аудио.

Решение

Добавьте элемент **audio** с атрибутом **src**, ссылающимся на аудиофайл, а также укажите резервный контент для старых браузеров:

```
<audio src="audio.ogg" controls>
    Скачать <a href="audio.ogg">42 эпизод "Учимся любить HTML5"</a>
</audio>
```

Если вы хотите, чтобы браузеры по умолчанию отображали интерфейс управления аудио, не забудьте добавить атрибут **controls** (рис. 4.1):

```
<audio src="audio.ogg" controls>
```

Аудиофайл в примере использует Ogg Vorbis (формат OGG) — бесплатный кодек с открытым исходным кодом (http://www.vorbis.com). Тем не менее существуют и другие аудиоформаты для Сети (табл. 4.1), и это одна из самых больших проблем реализации HTML5-аудио.

Кодек — это технология сжатия и распаковки данных. Аудиокодеки, используемые для сжатия и/или распаковки цифровых аудиоданных в различные форматы, позволяют сохранить высокое качество звука при минимальной скорости передачи информации.

Аудиокодеки

Спецификация HTML5 не дает четких рекомендаций по поводу использования конкретных аудиокодеков. Создателям браузеров было бы слишком трудно учесть все (http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2009-June/020620.html). Как видно из табл. 4.1, нет ни одного формата, который работает во всех браузерах.

Таблица 4.1. Текущая поддержка браузерами HTML5-аудиоформатов

Браузер	AAC (.aac)	MP3 (.mp3)	Ogg Vorbis (.ogg)	WAV (.wav)	WebM (.webm)
Chrome 6+		×	×		×
Firefox 3.6+			×	×	×
IE9+	×	×		×	
Opera 10.5+			×	×	
Safari 5+	×	×		×	

Рис. 4.1. По умолчанию в Chrome 9 панель управления аудио включает кнопку Воспроизведение/Пауза, индикатор выполнения и кнопку Регулировка/Отключение звука

Познакомимся с этими форматами поближе:

- AAC — формат сжатия с потерями, разработанный лучше, чем MP3, имеющий ту же скорость, но более высокое качество звука;
- MP3 — популярный и уже запатентованный формат, использующий сжатие с потерями и позволяющий уменьшить размер файла в десять раз;
- OGG — открытый альтернативный ресурс; как и MP3, использует формат сжатия с потерями;
- WAV — проприетарный аудиоформат без какого-либо сжатия;
- WEBM — открытый бесплатный медиаформат Google, основанный на аудиокодеке Vorbis.

Объединение нескольких источников

При выборе между форматами реальность такова, что, если вам необходимо обеспечить доступность контента для максимально широкой аудитории, придется сжать и включить в элемент **audio** *несколько* звуковых файлов. К счастью, HTML5 предоставляет такую возможность.

При использовании основного элемента audio атрибут src не нужен. Он необходим, только если вы ссылаетесь на один аудиоформат.

Оптимальный вариант — как минимум добавить файл в бесплатном формате OGG, *а также* в формате MP3 или WAV. Такой подход должен обеспечить поддержку аудио последними браузерами:

```
<audio controls>
    <source src="audio.ogg">
    <source src="audio.mp3">
    Скачать <a href="audio.ogg">42 эпизод "Учимся любить HTML5"</a>
</audio>
```

Предварительная загрузка аудио

Элемент **audio** имеет несколько атрибутов, позволяющих настроить воспроизведение звука.

Полное описание атрибутов, доступных для медиаэлементов HTML5, вы найдете в стандарте WHATWG: http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#media-element-attributes.

Атрибут **preload** сообщает браузеру, когда необходимо начать буферизацию аудио:

```
<audio controls preload>
```

Несмотря на то что **preload** частично поддерживается браузерами, его использование может пригодиться для оптимизации процесса загрузки. Достаточно указать этот атрибут и предоставить браузеру возможность самому выполнить соответствующие действия или указать один из трех вариантов.

- **preload="auto"** — браузер должен начать загрузку файла, но выбор остается за ним. При использовании мобильного устройства или при медленном соединении браузер может принять решение не начинать предварительную загрузку, чтобы сэкономить трафик.
- **preload="metadata"** — браузер самостоятельно не выполняет буферизацию аудио, пока пользователь не активизирует элементы управления. Но программа-обозреватель предварительно загружает метаданные, например треки и их продолжительность.
- **preload="none"** — аудио не будет загружаться, пока пользователь не активизирует элементы управления.

Обсуждение

Вдобавок к несоответствию форматов для разных браузеров сама поддержка аудио осуществляется несколько непоследовательно. У каждого браузера встречаются ошибки, причуды и странности, которые в ближайшем будущем, надеюсь, будут устранены производителями, но пока не стоит о них забывать.

На сайте 24Ways приводится список некоторых проблем браузеров: http://24ways.org/2010/the-state-of-html5-audio.

Создание резервного контента

Как видно из первого примера этого рецепта, элемент **audio** может включать *резервный контент*. Иными словами, если браузер не поддерживает HTML5-элемент **audio**, то пользователь увидит несколько видоизмененную информацию (рис. 4.2).

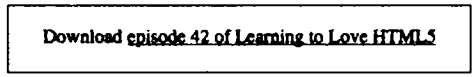

Рис. 4.2. Так отображается резервный контент в IE8, где отсутствует поддержка HTML5-элемента audio

Спецификация HTML5 говорит, что все дочерние элементы **audio**, кроме **source**, должны быть проигнорированы. Это означает, что для пользователей HTML5-совместимых браузеров предоставление резервного контента не приведет к негативным последствиям.

К примеру, можно добавить резервный Flash-проигрыватель:

```
<audio controls>
    <source src="audio.ogg">
    <source src="audio.mp3">
    <object data="player.swf?audio=audio.mp3">
        <param name="movie" value="player.swf?audio=audio.mp3">
        Видео и Flash не поддерживаются вашим браузером.
    </object>
</audio>
```

Или просто сообщить пользователю, что аудиофайл доступен для скачивания и может быть воспроизведен с помощью медиапроигрывателя (и наряду с этим ненавязчиво предложить перейти на новейший браузер):

```
<audio controls>
    <source src="audio.ogg">
    <source src="audio.mp3">
    Ваш браузер не поддерживает HTML5-аудио. Вам следует обновить его.
    Пока вы можете скачать <a href="audio.ogg">42 эпизод "Учимся любить
    HTML5"</a>.
</audio>
```

Доступные альтернативы

Еще одна проблема HTML5-аудио состоит в том, что для мультимедиа пока нет практической реализации альтернативного контента. *Теоретически*, доступ может открываться в два этапа: сначала авторы мультимедиа добавляют файл субтитров в контейнер формата OGV или MP3, а затем браузеры предоставляют пользователям интерфейс для доступа к этим субтитрам и подписям.

Использование атрибута alt для img нельзя назвать удачным решением. Этот вариант не предполагается в спецификации HTML5, и, что более важно, вспомогательные технологии не обрабатывают резервный контент audio подобным образом.

На данный момент можно рассмотреть несколько экспериментальных подходов:

- HTML5-видео с подписями, созданными в JavaScript: http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/;
- HTML5 и временное медиа: http://www.bbc.co.uk/blogs/rad/2009/08/html5.html;
- демонстрация HTML5-аудио и видео: http://www.annodex.net/~silvia/itext/.

Как вы заметили, некоторые из этих справочных ресурсов посвящены видео. Это произошло потому, что HTML5-аудио и видео настолько похожи, что и применяемые к ним подходы могут быть одинаково описаны (подробнее см. в главе 5).

У **audio** есть проблемы не только с субтитрами. Некоторые программы чтения с экрана вообще не признают этот элемент и просто пропускают его.

Кроме того, у браузеров нет единой поддержки работы **audio** через клавиатуру. В этом случае для HTML5 рекомендуется выполнить полное обновление: http://html5accessibility.com.

Право интеллектуальной собственности

Возможно, вы уже поняли, что HTML5-элемент **audio** — не универсальное решение. Но это связано не только с необходимостью хранения аудио в различных форматах или с отсутствием поддержки браузерами — проблема еще в том, что HTML5 не обеспечивает защиту от копирования.

Содержимое элемента **audio** (и **video**, см. в главе 5) можно легко сохранить на жестком диске пользователя подобно **img**, и, следовательно, такое решение иногда неприемлемо. Если необходимо предусмотреть технические средства защиты авторских прав, то лучшим решением сегодня будет использование плагинов, а не **audio**.

Дополнительная информация

Для получения более подробной информации о доступности HTML5-мультимедиа W3C предлагает глубже рассмотреть известные проблемы и возможные решения: http://www.w3.org/html/wg/wiki/MultimediaAccessibilty.

4.2. Управление аудиопотоком

Проблема

Необходимо иметь возможность управлять воспроизведением HTML5-аудио в браузере.

Решение

В спецификации для элемента **audio** предусмотрено несколько атрибутов, позволяющих просто и быстро контролировать воспроизведение звука.

- **autoplay** — сообщает браузеру о том, что необходимо начать воспроизведение аудио, как только будет загружена страница.

Я не решаюсь даже упоминать этот атрибут, поскольку он делает именно то, что больше всего раздражает на сайтах (http://www.punkchip.com/2009/04/autoplay-is-bad-for-all-users). Таким образом, пока autoplay все еще существует, не используйте его. Серьезно. Не надо.

- **loop** — еще один описательный атрибут. Он указывает браузеру циклично воспроизводить аудио.

Он не так ужасен, как атрибут autoplay, но все же применяйте его с осторожностью.

Как и **controls**, атрибуты **autoplay** и **loop** — логические, поэтому при необходимости достаточно включить их в открывающий тег **audio**:

```
<audio controls loop>
```

Обсуждение

Что делать, если нужно получить больше возможностей управления, чем могут предоставить основные атрибуты? К счастью, у **audio** и **video** есть атрибуты, события и методы для работы с JavaScript, позволяющие создавать пользовательские элементы управления, в том числе:

- **canplaytype(*mun*)** — возвращает строку, указывающую, может ли браузер воспроизводить данный тип медиафайла;
- **currentTime** — определяет текущую позицию воспроизведения, заданную в секундах;
- **duration** — содержит длину аудиофайла в секундах;
- **play();** — начинает воспроизведение с текущей позиции;
- **pause();** — приостанавливает воспроизведение аудио, если оно проигрывается.

Предположим, вам необходимо добавить элементы управления, позволяющие пользователю перейти к определенному моменту аудиофайла. Вы можете добавить эту функцию, разместив кнопку и написав JavaScript-код для управления методом **play()** на основе чтения/записи свойства **currentTime**:

```
<audio>
    <source src="audio.ogg">
    <source src="audio.mp3">
</audio>
<button title="Воспроизвести с 30 секунды" onclick="playAt(30);">
30 секунда</button>
<script>
    function playAt(seconds){
        var audio = document.getElementsByTagName("audio")[0];
        audio.currentTime = seconds;
        audio.play();
    }
</script>
```

Метода **stop** не существует, но можно имитировать его функциональность, применив подход из предыдущего примера. Для этого введем метод **pause()**, чтобы вернуться к началу аудиофайла, используя **currentTime**:

```
<audio>
    <source src="audio.ogg">
    <source src="audio.mp3">
</audio>
<button title="Воспроизвести с 30 секунды" onclick="playAt(30);">
30 секунда</button>
<button title="Остановить воспроизведение" onclick="stopAudio();">
Стоп</button>
<script>
    function playAt(seconds){
        var audio = document.getElementsByTagName("audio")[0];
        audio.currentTime = seconds;
        audio.play();
    }
    function stopAudio(){
        var audio = document.getElementsByTagName("audio")[0];
        audio.currentTime = 0;
        audio.pause();
    }
</script>
```

При создании собственных пользовательских элементов управления не используйте для audio логический атрибут controls.

Для получения дополнительной информации о создании пользовательских элементов управления обратитесь к рецепту 4.5.

Дополнительная информация

Для того чтобы иметь минимальное представление о воспроизведении звука с пользовательскими элементами управления для Opera, прочитайте статью

«Все, что нужно знать о HTML5-видео и аудио»: http://dev.opera.com/articles/view/everything-you-need-to-know-about-html5-video-and-audio/.

4.3. Создание <audio> с помощью JavaScript

Проблема

Необходимо в реальном режиме времени создавать аудио на веб-странице.

Решение

С помощью методов, определенных Mozilla Audio Data API, можно создавать в браузере аудио без атрибутов **src** или элементов **source** (https://wiki.mozilla.org/Audio_Data_API#Writing_Audio):

- **mozSetup(*каналы, частоты_дискретизации*)** — определяет каналы и частоты дискретизации для созданного звукового потока;
- **mozWriteAudio(*буфер*)** — записывает фрагменты из созданного для аудио массива;
- **mozCurrentSampleOffset()** — возвращает текущую позицию воспроизведения аудио, заданную во фрагментах.

Обсуждение

Такая реализация **audio** возможна далеко не во всех браузерах. В настоящее время этот элемент поддерживают только Firefox 4+ и Chrome Beta. Это скорее экспериментальный подход, нежели решение, предназначенное для основного использования.

Если вам хочется ознакомиться с экспериментальными типами, посмотрите короткую видеопрезентацию о возможностях Mozilla Audio Data API: http://www.youtube.com/watch?v=1Uw0CrQdYYg.

Дополнительная информация

На Barcamp London 8 приводится статья «jasmid — MIDI, соединенное с JavaScript и HTML5-аудио», рассматриваются проблемы и практические последствия динамического создания аудио в браузере: http://matt.west.co.tt/music/jasmid-midi-synthesis-with-javascript-and-html5-audio/.

4.4. Визуализация <audio> с помощью <canvas>

Проблема

Необходимо визуализировать элемент **audio** с помощью **canvas**.

Решение

В этом примере продемонстрирована реализация элемента **canvas**, визуализирующего аудиосигнал (рис. 4.3):

```
<audio src="audio.ogg"></audio>
<canvas width="512" height="100"></canvas>
<button title="Сгенерировать сигнал" onclick="genWave();">
Сгенерировать сигнал</button>
<script>
    function genWave(){
        var audio = document.getElementsByTagName("audio")[0];
        var canvas = document.getElementsByTagName("canvas")[0];
        var context = canvas.getContext('2d');
        audio.addEventListener("MozAudioAvailable", buildWave, false);
        function buildWave (event){
            var channels = audio.mozChannels;
            var frameBufferLength = audio.mozFrameBufferLength;
            var fbData = event.frameBuffer;
            var steplnc = (frameBufferLength / channels) / canvas.width;
            var waveAmp = canvas.height / 2;
            canvas.width = canvas.width;
            context.beginPath();
            context.moveTo(o, waveAmp - fbData[o] * waveAmp);
            for(var i=l; i < canvas.width; i++){
                context.lineTo(i, waveAmp - fbData[i*stepInc] * waveAmp);
            }
            context.strokeStyle = "#fff";
            context.strokeQ;
        }
        audio.play();
    }
</script>
```

В этом решении комбинируются некоторые методы Mozilla Audio Data API и элемент **canvas**, рассматриваемый в главе 9. Разберем пример по частям, начиная с ключевых элементов **audio**, **canvas** и элемента **button**, запускающего визуализацию:

```
<audio src="audio.ogg"></audio>
<canvas width="512" height="100"></canvas>
<button title=" Сгенерировать сигнал" onclick="genWave();">
Сгенерировать сигнал</button>
```

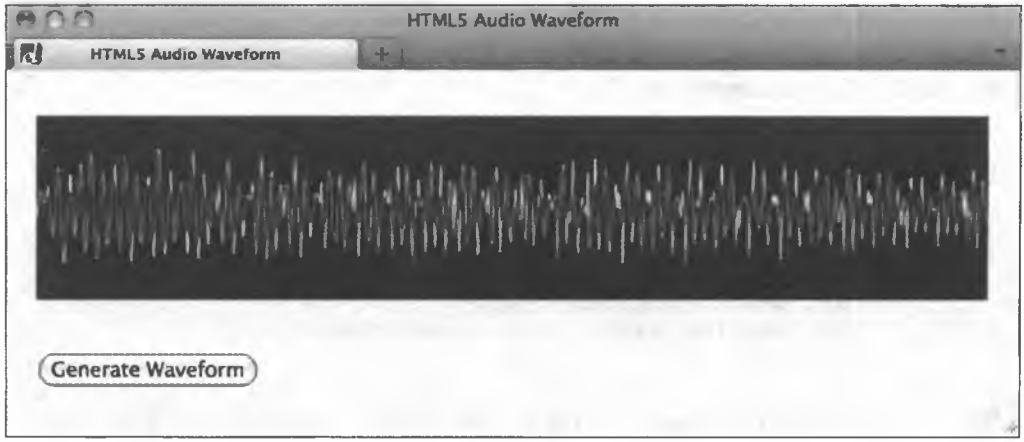

Рис. 4.3. Аудиосигнал, визуализированный с помощью элемента canvas в Firefox 5

В примере для упрощения в audio используется атрибут src, но можно указать несколько элементов source.

С помощью CSS добавьте цвет фона для простейшего отображения **canvas**:

```
<style>
    canvas {background: #000;}
</style>
```

Обратите внимание, что значения width и height в canvas — атрибуты DOM, а не атрибуты стиля. Таким образом, чтобы браузер знал размеры области для рисования, необходимо указать их в разметке, а не в CSS. Подробнее о canvas читайте в главе 9.

Теперь о JavaScript. Во-первых, создайте основную функцию для генерации волн:

```
<script>
    function genWave(){
```

Внутри этой функции получите элементы **audio** и **canvas**:

```
        var audio = document.getElementsByTagName("audio")[0];
        var canvas = document.getElementsByTagName("canvas")[0];
```

Затем установите для **canvas** контекст рисования (http://diveintohtml5.info/canvas.html#shapes):

```
        var context = canvas.getContext('2d');
```

Далее с помощью методов Mozilla Audio Data API добавьте слушателя для события сбора данных об аудиофайле (https://developer.mozilla.org/en/Introducing_the_Audio_API_Extension#section_2):

```
audio.addEventListener("MozAudioAvailable", buildWave, false);
```

Добавьте закрытые функции для построения волны в **canvas** и получите значения **channels** и **frameBufferLength**:

```
function buildWave (event){
    var channels = audio.mozChannels;
    var frameBufferLength = audio.mozFrameBufferLength;
```

Обратите внимание, что нужно отделить frameBufferLength от channels, поскольку frameBuffer содержит массив из фрагментов, которые не отделяются от каналов, а передаются вместе.

Получите данные **frameBuffer**:

```
    var fbData = event.frameBuffer;
```

Задайте шаг приращения:

```
    var stepInc = (frameBufferLength / channels) / canvas.width;
```

и установите амплитуду волны:

```
    var waveAmp = canvas.height / 2;
```

Затем сбросьте **canvas** так, чтобы кривые не накладывались друг на друга:

```
    canvas.width = canvas.width;
```

Наконец, постройте кривую и задайте ее свойства:

```
    context.beginPath();
    context.moveTo(0, waveAmp - fbData[0] * waveAmp);
    for(var i=1; i < canvas.width; i++){
        context.lineTo(i, waveAmp - fbData[i*stepInc] * waveAmp);
    }
    context.strokeStyle = "#fff";
```

Добавьте кривую на **canvas**:

```
    context.stroke();
}
```

и воспроизводите аудио:

```
    audio.play();
}
</script>
```

Обсуждение

Подобно ограничениям при динамическом создании аудио с помощью JavaScript (см. рецепт 4.3) для Firefox 4+, существуют ограничения и для такого метода визуализации **audio** с помощью **canvas**. Это связано с тем, что только Mozilla Audio Data API позволяет получить доступ к ключевым звуковым данным (в нашем примере — **frameBuffer**), необходимым для создания рисунка **canvas**.

Кроме того, такой метод визуализации звука должен работать на веб-сервере, и в связи с мерами безопасности Firefox он требует, чтобы аудиофайл находился на том же сервере (https://wiki.mozilla.org/Audio_Data_API#Security).

Хотя этот рецепт и использует Mozilla Audio Data API, браузеры (включая Firefox) поддерживают Web Audio API от W3C. Для получения дополнительной информации читайте спецификацию на https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html.

Дополнительная информация

В проекте 9elements можно найти более сложное и интерактивное применение **audio** и **canvas**: http://9elements.com/io/projects/html5/canvas/.

4.5. Создание аудиопроигрывателя

Проблема

Необходимо создать аудиопроигрыватель.

Решение

Чтобы решить поставленную задачу, расширим концепцию из рецепта 4.2 и не станем по умолчанию полагаться на атрибут **controls** (рис. 4.4).

Определение элемента audio и структуры проигрывателя

Начнем с разметки для HTML5-элемента **audio** и элементов управления проигрывателем:

```
<div id="player">
    <audio src="media/audio.mp3"></audio>

    <div class="playerControls">
        <button id="audioPlay" title="Воспроизведение"
            onclick="playPause();">&#x25ba;</button>
        <button id="audioPause" class="hidden" title="Пауза"
            onclick="playPause();">&#x2590;&#x2590;</button>
        <button id="audioStop" title="Стоп"
            onclick="playStop();">&#x25a0;</button>

        <div id="audioSeek">
            <div id="audioLoaded"></div>
        </div>

        <ul id="audioTimes">
            <li id="audioElapsed">00:00:00</li>
            <li id="audioDuration">00:00:00</li>
        </ul>
    </div>
</div>
```

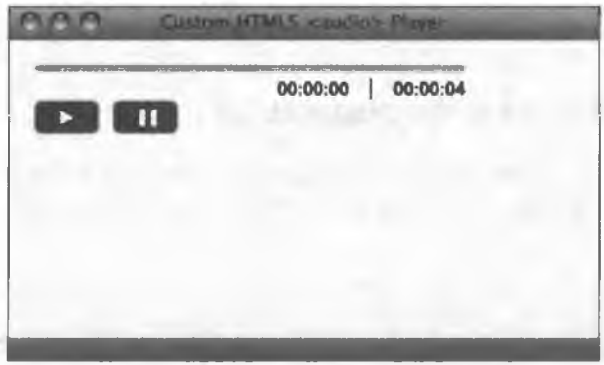

Рис. 4.4. Окончательный вид пользовательского аудиопроигрывателя в Safari 5

Для простоты в этом примере используем только переключатель Воспроизведение/Пауза, кнопку Стоп и индикатор выполнения.

Выбор элемента button для управления проигрывателем оптимален с точки зрения семантики и доступности, но не является строго обязательным.

JavaScript API

Теперь перейдем к мощному механизму DOM API, обеспечивающему функциональность разметки проигрывателя.

Для повышения удобства и доступности можно определить наличие поддержки HTML5-элемента audio (http://diveintohtml5.info/everything.html). Если он поддерживается, то будет добавлен, а также начнет применяться разметка проигрывателя. Это означает, что браузеры, которые не поддерживают audio, не смогут отобразить такой проигрыватель. Modernizr (http://modernizr.com) может помочь обнаружить подобную проблему.

Во-первых, объявим элемент **audio** и элементы управления проигрывателем и убедимся, что сценарий появляется в исходном коде сразу после элемента **audio**:

```
<script>
    audio = document.getElementsByTagName("audio")[0];
    audioDuration = document.getElementById("audioDuration");
    audioElapsed = document.getElementById("audioElapsed");
    audioPlay = document.getElementById("audioPlay");
    audioPause = document.getElementById("audioPause");
    audioStop = document.getElementById("audioStop");
    audioLoaded = document.getElementById("audioLoaded");
```

Затем укажем полную информацию об аудиофайле:

```
    audio.addEventListener("loadedmetadata", setDuration, false);
    audio.addEventListener("timeupdate", setElapsed, false);
```

Далее определим функции перемещения индикатора выполнения:

```
    function setDuration(event) {
        audioDuration.innerHTML = timeFormatter(audio.duration);
    }

    function setElapsed(event) {
        audioElapsed.innerHTML = timeFormatter(audio.currentTime);
        amountLoaded = (audio.currentTime/audio.duration)*100;
        audioLoaded.style.width = amountLoaded + 'px';
    }
```

И функцию переключения Воспроизведение/Пауза:

```
    function playPause() {
        if (audio.paused){
            audio.play();
            audioPlay.className = 'hidden';
            audioPause.className = '';
        } else {
            audio.pause();
            audioPlay.className = '';
            audioPause.className = 'hidden';
        }
    }
```

Затем зададим функцию остановки, основанную на материале из рецепта 4.2:

```
    function playStop() {
        audio.pause();
```

продолжение ⇩

```
        audio.currentTime=0;
        audioPlay.className = '';
        audioPause.className = 'hidden';
    }
```

и функцию для отображения времени на индикаторе выполнения:

```
    function timeFormatter(seconds){
        function zeroPad(str) {
            if (str.length > 2) return str;
            for (i=0; i<(2-str.length); i++) {
                str = "0" + str;
            }
            return str;
        }
        var minute = 60,
        hour = minute * 60,
        hStr = "",
        mStr = "",
        sStr = "";

        var h = Math.floor(seconds / hour);
        hStr = zeroPad(String(h));

        var m = Math.floor((seconds - (h * hour)) / minute);
        mStr = zeroPad(String(m));

        var s = Math.floor((seconds - (h * hour)) - (m * minute));
        sStr = zeroPad(String(s));

        return (hStr + ":" + mStr + ":" + sStr);
    }
</script>
```

Стили CSS

Наконец, определим стиль проигрывателя, чтобы он не выглядел совсем простым (рис. 4.5). Начнем с размеров окна и свойств отображения кнопок **button**:

```
#player {
    height: 50px;
    padding: 10px;
    position:relative;
    width: 300px; }

button {
    background: #666;
    border:1px;
    -moz-border-radius: 5px;
    border-radius: 5px;
    bottom: 10px;
    color: #fff;
```

```
    padding: 5px;
    position:absolute;
    width:45px;
}

#audioStop {
    font-size:22px;
    left: 65px;
    line-height: 11px;
}
```

Затем добавим стили для переключателя Воспроизведение/Пауза:

```
#audioPlay.hidden,
#audioPause.hidden { display:none; }

#audioSeek {
    background: #ccc;
    border: 1px solid #000;
    -moz-border-radius: 10px;
    border-radius: 10px;
    display:block;
    height:2px;
}
```

и стили для индикатора выполнения:

```
#audioLoaded {
    background: #0c0;
    border: 1px solid #0c0;
    -moz-border-radius: 10px;
    border-radius: 10px;
    display:block;
    height:1px;
}
```

И наконец, стили для секундомера:

```
#audioTimes {
    float:right;
    list-style:none;
    margin: 5px 0 0;
}

#audioTimes li {
    font:bold 13px Arial, Helvetica sans-serif;
    float: left;
}

#audioTimes li:first-child {
    border-right: 1px solid #000;
    margin-right: 15px;
    padding-right: 15px;
}
```

Рис. 4.5. Аудиопроигрыватель без использования стилей в Safari 5

Обсуждение

Ваш проигрыватель не работает? Есть несколько готовых решений с пользовательской панелью управления, включающей даже список воспроизведения:

- плагин jPlayer JQuery: http://www.jplayer.org;
- MooTools HTML5 Audio Player: http://simulacre.org/mootools-html5-audio-player.

Дополнительная информация

Руководство «Создание собственного доступного HTML5-медиа-проигрывателя»: http://terrillthompson.blogspot.com/2010/08/creating-your-own-accessible-html5.html.

5 Встроенное видео

Эмили Льюис

5.0. Введение

Новые встроенные HTML5-элементы предоставляют дизайнерам и разработчикам больше возможностей для использования на сайтах мультимедиа.

Элемент **video** обладает теми же атрибутами, что и **audio**, и имеет аналогичный синтаксис. Кроме того, можно стилизовать его и манипулировать им с помощью CSS и JavaScript. Так же, как **audio**, элементу **video** свойственны некоторые проблемы реализации.

5.1. Добавление HTML5-видео

Проблема

Необходимо воспроизводить на веб-странице встроенное видео.

Решение

Используйте элемент **video** с атрибутом **src**, ссылающимся на видеофайл:

```
<video src="video.ogv"></video>
```

Для отображения по умолчанию панели управления видеопроигрывателем добавьте логический атрибут **controls** (рис. 5.1):

```
<video src="video.ogv" controls></video>
```

Рис. 5.1. Панель управления, устанавливаемая по умолчанию для видеопроигрывателя в Firefox 4

Панель управления проигрывателем в разных браузерах может выглядеть по-разному. С помощью JavaScript и CSS установите для элементов управления собственные стили (см. рецепт 5.6).

Предварительная загрузка

Атрибут **preload** *сообщает* браузеру, когда и как следует начать загрузку видео:

```
<video src="video.ogv" controls preload></controls>
```

Можно пропустить этот атрибут и оставить выбор за программой или определить браузеру поведение (подробнее см. рецепт 4.1):

- **preload="auto"** или просто **preload**;
- **preload="metadata"**;
- **preload="none"**.

Резервный контент

Возьмите за правило добавлять *резервный контент* для **video**:

```
<video src="video.ogv" controls>
    Ваш браузер не поддерживает HTML5-видео. <a href="video.ogg">Скачать
    вступительный ролик "Учимся любить HTML5"</</a>.
</video>
```

Предоставление резервного контента — это не то же самое, что обеспечение доступности или добавление открытой информации.

Резервный контент отображается в браузерах, не поддерживающих видео. Оно решает две задачи: информирует пользователя о том, что его браузер не поддерживает HTML5-видео, и предоставляет ссылку для загрузки видео.

Хорошая статья о текущем уровне доступности HTML5-видео находится по адресу http://john.foliot.ca/accessibility-and-html5-today/.

Наряду с различными форматами видео в качестве запасного варианта можно добавить Flash-видео.

Из-за отсутствия достаточной поддержки можно помещать резервный контент не только как элемент **video**, но и как **object**:

```
<video src="video.ogv" controls>
    <object type="application/x-shockwave-flash"
            data="player.swf?file="video.mp4">
        <param name="movie" value="player.swf?file="video.mp4">
        <a href="video.mp4"> Скачать вступительный ролик "Учимся любить
            HTML5"</a>
    </object>
</video>
```

Обсуждение

К сожалению, воспроизведение видео не так просто реализовать, как это может показаться, поскольку не все браузеры поддерживают один и тот же набор видеоформатов.

Видеокодеки

Как и в случае с **audio** (см. рецепт 4.1), спецификация HTML5 не уточняет, какие именно видеокодеки нужно использовать. Какой формат поддерживать, решают разработчики браузеров (табл. 5.1). Это нормально, но согласовать каждый случай для всех браузеров практически невозможно. Кроме того, трудно выбрать один видеоформат, поддерживать который нужно больше остальных. Для отображения в наиболее широком диапазоне браузеров видео должно быть сжато и опубликовано в нескольких форматах, что позволит избежать описанной ситуации.

В табл. 5.1 У означает «удалено», С означает «необходимо скачать», а Д — «да (поддерживается)». Заметим, что в браузере Google Chrome версии 11 отсутствует поддержка H.264. Браузер IE9 поддерживает VP8, но только в том случае, если пользователь устанавливает эти кодеки в Windows.

Таблица 5.1. Текущая поддержка браузерами форматов видео

Браузер	H.264 (.mp4)	Ogg Theora (.ogv)	VP8 (.webm)
Chrome 6+	У	Д	Д
Firefox 3.6+		Д	Д
IE9+	Д		С
Opera 10.5+		Д	Д
Safari 5+	Д		

Как видно из табл. 5.1, если необходимо обеспечить воспроизведение видео во всех браузерах, нужно добавить хотя бы форматы MP4 и OGV. Рассмотрим некоторые поддерживаемые форматы.

- MP4 — формат-контейнер для проприетарного кодека H.264, кодирующий видео, в том числе высокой четкости, для полного спектра устройств.
- OGV — свободный формат-контейнер с открытым исходным кодом для кодека Theora. Результаты сжатия имеют более низкое качество, чем H.264.
- WEBM — еще один формат-контейнер с открытым исходным кодом, использующий новый бесплатный кодек VP8 от Google.

Google добавил в Chrome поддержку H.264 (http://blog.chromium.org/2011/01/html-video-codec-support-in-chrome.html), а тем временем VP8 поддерживается браузером IE9 с установленным декодером (http://windowsteamblog.com/windows/b/bloggingwindows/archive/2010/05/19/another-follow-up-on-html5-video-in-ie9.aspx).

Почему возникает такая неприятная ситуация с видеокодеками? Огромную роль играет соблюдение прав интеллектуальной собственности и лицензионные сборы. Производители браузеров, выбирающие определенные форматы (и, следовательно, кодеки), изначально обладают правами на соответствующую интеллектуальную собственность.

Apple и Microsoft оплатили лицензию на использование в своих браузерах видеокодека H.264. Тем временем создатели браузеров Firefox и Opera предпочитают поддерживать свободные форматы с открытым кодом.

Для того чтобы разобраться, почему существует так много видеокодеков и форматов, посмотрите вступительный ролик Стефани Салливан Рэвис (Stephanie Sullivan Rewis) о HTML5-видео: http://tv.adobe.com/watch/adc-presents/videoandhtml5part2gettingstarted/. Но есть один нюанс: чтобы посмотреть это видео, необходимо установить Flash для браузера.

Поддержка браузеров

Наряду с несоответствием некоторых кодеков и форматов-контейнеров в разных браузерах сама поддержка видео осуществляется частично. Элемент **video** не поддерживается в Internet Explorer версии 8 и ниже, и в этих случаях необходимо определять резервный контент.

Дополнительная информация

Ресурс о разработке видео и экспериментах с поддержкой HTML5-видео, найденный в интернет-архиве: http://www.archive.org/details/movies.

5.2. Мультибраузерная поддержка видео

Проблема

Необходимо убедиться, что встроенное видео будет воспроизводиться популярными браузерами.

Решение

Используйте для **video** дочерний элемент **source**, указав формат каждого видеофайла:

```
<video controls>
    <source src="video.mp4" />
    <source src="video.ogv" />
    Ваш браузер не поддерживает HTML5-видео.
</video>
```

Обратите внимание, что из элемента video необходимо убрать атрибут src вместе со значением, как показано в рецепте 5.1.

Как видно из табл. 5.1, все браузеры поддерживают HTML5-видео в форматах MP4 и OGV. Но задача остается прежней. Перечислите в **source** все форматы, начиная с наиболее предпочтительных, так как браузеры станут воспроизводить видео, найдя первый поддерживаемый формат.

На сайте Video for Everybody (Видео для всех) подробно описаны ошибки и особенности работы с HTML5-видео: http://camendesign.com/code/video_for_everybody/.

Обсуждение

Возьмите за правило не только перечислять сами видеофайлы, но и указывать MIME-тип для каждого из них:

```
<source src="video.mp4" type="video/mp4" />
<source src="video.ogv" type="video/ogg" />
```

Firefox не воспроизводит Ogg-видео, если не указан MIME-тип.

Поскольку форматы видеофайлов — это просто контейнеры для различных кодеков, необходимо также указать кодек, используемый для сжатия:

```
<source src="video.ogv" type="video/ogg; codecs='theora'">
```

Наличие подобных данных сокращает время на обработку видео браузером, так как ему не требуется загружать файл, чтобы самостоятельно определить его формат.

Вот, к примеру, некоторые MIME-типы кодеков:

```
type="video/ogg; codecs='theora, vorbis'"
type="video/mp4; codecs='avc1.42E01E'"
type="video/webm; codecs='vp8, vorbis'"
```

К сожалению, даже если вы объявите в HTML MIME-тип с соответствующим кодеком, все равно могут возникнуть проблемы. Если на *сервере* не настроена поддержка MIME-типов, используемых вашим видео, могут быть ошибки при его воспроизведении.

Однако эту поддержку можно настроить. Информация для медиаформата OGG на Mozilla Developer Center (https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media) в равной степени применима для WEBM и MP4.

Дополнительная информация

Если у вас есть собственное видео, которые вы хотите конвертировать в другие форматы, воспользуйтесь следующими инструментами:

- Miro VideoConverter: http://www.mirovideoconverter.com/;
- WebM Tools: http://www.webmproject.org/tools/;
- MPEG StreamClip: http://www.squared5.com/;
- TinyOgg: http://tinyogg.com/;
- OggConvert: http://oggconvert.tristanb.net/.

5.3. Настройка видеоразрешения

Проблема

Необходимо установить ширину и высоту окна видеопроигрывателя.

Решение

Добавьте в **video** атрибуты **width** и **height**, указав для них необходимые значения:

```
<video controls width="640" height="360">
    <source src="video.mp4" type="video/mp4" />
    <source src="video.ogv" type="video/ogg" />
    Ваш браузер не поддерживает HTML5-видео.
</video>
```

Обсуждение

Браузеры отображают видео согласно указанным размерам проигрывателя, а не в соответствии с фактическим разрешением видео. Подобное несоответствие размеров может привести к потере качества, ведь расширение окна видеопроигрывателя может превосходить собственное разрешение видео, или видео с высоким разрешением может просто не вписаться в небольшое окно проигрывателя.

Таким образом, если это возможно, используйте для элемента **video** *то же* разрешение, какое имеет видеофайл, и не изменяйте размеры видео в соответствии с **width** и **height**.

Не следует устанавливать ширину и высоту элементов **video** в браузере по умолчанию в зависимости от разрешения видеофайла. Однако если размеры не указаны, браузер не сможет определить, сколько места необходимо для видео, и, вероятно, после загрузки видеоданных придется перезапускать страницу.

В CSS значения **width** и **height** должны быть заданы в пикселах как целые числа, а не в процентах или других единицах. Указание единиц измерения помогает браузерам более эффективно отображать страницы, что обычно очень ценится пользователями.

Не применяйте **width** и **height** для изменения текущего разрешения видео, а воспользуйтесь специально предусмотренным программным обеспечением.

Дополнительная информация

Статья «Погружение в HTML5» Марка Пилгрима (http://diveintohtml5.info/video.html) содержит довольно подробные инструкции по применению множества инструментов для сжатия, кодировки и пикселизации: http://www.emdpi.com/csspixel.html.

5.4. Отображение замещающего изображения до воспроизведения видео

Проблема

До того как пользователь запустит воспроизведение, необходимо отображать кадр из видео или изображение-заставку.

Решение

Добавьте атрибут **poster**, используя в качестве значения путь к файлу с замещающим изображением:

```
<video controls width="640" height="360" poster="video_still.png">
    <source src="video.mp4" type="video/mp4" />
    <source src="video.ogv" type="video/ogg" />
    Ваш браузер не поддерживает HTML5-видео.
</video>
```

Обсуждение

Изображение-заставка может пригодиться пользователю, когда видео не загружается или не может быть воспроизведено. Например, как в случае

с просмотром в Firefox MP4-видео, не имеющего резервного контента (рис. 5.2).

Рис. 5.2. Браузер Firefox, не поддерживающий MP4-файлы, показывает темно-серый квадрат, если изображение-заставка не задано

К сожалению, различные браузеры по-разному обрабатывают изображения-заставки.

- Firefox растягивает изображение по размеру окна проигрывателя (см. рецепт 5.3).
- Chrome сохраняет оригинальный размер изображения.
- Safari показывает изображение-заставку, используемое до начала загрузки видеоданных, в виде первого кадра видео. Если размер изображения-заставки не соответствует разрешению видео, то это разрешение будет изменено в соответствии с размером изображения (рис. 5.3).

Дополнительная информация

В своем блоге Джон Фолиот (John Foliot) указывает на неполную доступность атрибута **poster**: http://john.foliot.ca/the-current-problem-with-the-poster-attribute-of-the-video-element/. Имейте в виду, что при использовании **poster** существуют некоторые ограничения.

Рис. 5.3. Safari 5 изменяет разрешение видео в соответствии с размером изображения-заставки, поэтому важно использовать подходящее изображение

5.5. Цикличное воспроизведение видео

Проблема

Необходимо обеспечить автоматическое цикличное воспроизведение видео.

Решение

Добавьте элементу **video** логический атрибут **loop**:

```
<video controls width="640" height="360" loop>
    <source src="video.mp4" type="video/mp4" />
    <source src="video.ogv" type="video/ogg" />
    Ваш браузер не поддерживает HTML5-видео.
</video>
```

Обсуждение

На момент написания этой статьи Firefox не поддерживал **loop** без установки дополнения Media Loop (https://addons.mozilla.org/en-US/firefox/addon/media-loop-45730/). Остальные браузеры изначально поддерживают цикличное воспроизведение.

Тем не менее используйте **loop** *с осторожностью*. Подумайте, как пользователи отнесутся к бесконечно зацикленному видео, будет ли это уместно?

Атрибут autoplay. Возможно, будет полезно рассмотреть еще один логический атрибут элемента **video** — **autoplay**. Аналогично его применению с элементом **audio** (см. рецепт 4.2), в этом случае воспроизведение видео начинается сразу после загрузки страницы, что может вызвать недовольство пользователей.

Дополнительная информация

Не все пользователи Firefox знают о дополнении Media Loop и тем более о необходимости его установки. Если вам чрезвычайно важно обеспечить цикличное воспроизведение видео во всех браузерах, рассмотрите решение на основе JavaScript: http://forestmist.org/2010/04/html5-audio-loops/.

5.6. Управление видео с помощью <canvas>

Проблема

Необходимо обеспечить возможность встроить видео с YouTube, выбрав лучший вариант превью, нежели вариант по умолчанию (рис. 5.4).

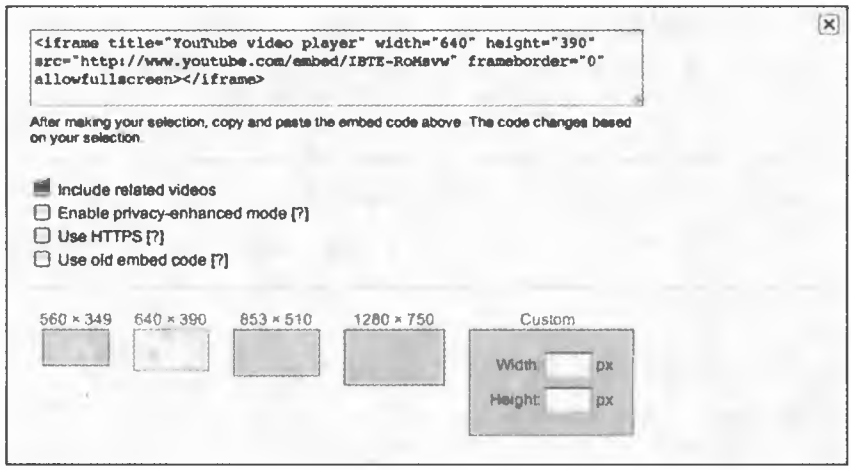

Рис. 5.4. Для выбора размеров встроенного видео YouTube предлагает набор статических изображений

Решение

В этом примере используется элемент **canvas** (см. главу 9), который обеспечивает лучший и более уместный вариант выбора превью, позволяющей увидеть, как видео будет отображаться в разном разрешении (рис. 5.5).

Рис. 5.5. Благодаря canvas и масштабированию в Firefox прорисован один и тот же видеокадр разного размера

Добавление тегов <video> и <canvas>

Для начала возьмите ранее описанную в этой главе разметку для элемента **video** и добавьте кнопку **button**, а также два элемента **canvas**: для просмотра превью среднего и маленького размера.

Каждому элементу присвойте уникальный **id**:

```
<video id="origVideo" width="640" height="360">
    <source src="video.mp4" type="video/mp4" />
    <source src="video.ogv" type="video/ogg" />
    Ваш браузер не поддерживает HTML5-видео.
</video>
```

```
<button title="Создать превью" onclick="generatePreview();">
    Создать превью</button>

<canvas id="previewMed"></canvas>
<canvas id="previewSm"></canvas>
```

Создание превью

Создайте функцию получения превью и определите с помощью JavaScript переменные для элементов **video** и **canvas**:

```
<script>
    function generatePreview(){
        var video = document.getElementById('origVideo');
        var canvas1 = document.getElementById('previewMed');
        var context1 = canvas1.getContext('2d');
        var canvas2 = document.getElementById('previewsSm');
        var context2 = canvas2.getContext('2d');
```

Переменные контекста 2D необходимо задавать для каждого элемента **canvas** отдельно.

Определение размеров превью

Далее в этой же функции определите размеры для каждого рисунка **canvas**:

```
        canvas1.width = 320;
        canvas1.height = 180;
        canvas2.width = 160;
        canvas2.height = 90;
```

Добавление слушателя события

Затем добавьте слушателя для события для вызова рисования **canvas** во время воспроизведения видео:

```
        video.addEventListener('play', function(){
            drawVideo(this,context1,context2);
        },false);
```

и закончите функцию:

```
        video.play();
    }
```

Рисование превью

Наконец, добавьте функцию, рисующую превью в каждом элементе **canvas**:

```
    function drawVideo(video,canvas1,canvas2) {
        if(video.paused || video.ended) return false;
```

продолжение ⇩

```
        canvas1.drawImage(video,0,0,320,180);
        canvas2.drawImage(video,0,0,160,90);
        setTimeout(drawVideos,25,video,canvas1,canvas2);
    }
</script>
```

Эта функция сначала проверяет, воспроизводится ли видео:

```
if(video.paused || video.ended) return false;
```

Если воспроизводится, то для масштабирования изображения **canvas**, в два раза уменьшенного по отношению к оригинальному видео, используется **drawImage**:

```
canvas1.drawImage(video,0,0,320,180);
```

и для второго **canvas**, который в два раза меньше **previewMed**:

```
canvas2.drawImage(video,0,0,160,90);
```

Вот пример параметров масштабирования:

```
object.drawImage(источник,x,y,ширина,высота);
```

Здесь *x* и *y* — координаты верхнего левого угла изображения выбранного **canvas**, а *ширина* и *высота* — размеры изображения данного элемента **canvas**.

Наконец, укажите, что необходимо вызывать **drawPreviews** каждые 25 мс, что примерно составляет 40 кадров в секунду (1000 мс / 40 фпс = 25):

```
setTimeout(drawVideos,25,video,canvas1,canvas2);
```

Обсуждение

Разметка и сценарий для этого примера выглядят следующим образом:

```
<video id="origVideo" width="640" height="360">
    <source src="video.mp4" type="video/mp4" />
    <source src="video.ogv" type="video/ogg" />
    Ваш браузер не поддерживает HTML5-видео.
</video>

<button title="Создать превью" onclick="generatePreview();">
    Создать превью</button>

<canvas id="previewMed"></canvas>
<canvas id="previewSm"></canvas>

<script>
    function generatePreview(){
        var video = document.getElementById('origVideo');
```

```
        var canvas1 = document.getElementById('previewMed');
        var context1 = canvas1.getContext('2d');
        var canvas2 = document.getElementById('previewsSm');
        var context2 = canvas2.getContext('2d');

        canvas1.width = 320;
        canvas1.height = 180;
        canvas2.width = 160;
        canvas2.height = 90;

        video.addEventListener('play', function(){
            drawVideo(this,context1,context2);
        },false);

        video.play();
    }

    function drawVideo(video,canvas1,canvas2) {
        if(video.paused || video.ended) return false;
        canvas1.drawImage(video,0,0,320,180);
        canvas2.drawImage(video,0,0,160,90);
        setTimeout(drawVideos,25,video,canvas1,canvas2);
    }
</script>
```

Теперь благодаря встроенному HTML5-видео веб-разработчики могут погрузиться в мир видео и усовершенствовать уже имеющиеся достижения когда-то запатентованных технологий.

Дополнительная информация

HTML5 Doctor предлагает демо-версии и подробные описания других способов совместного использования **canvas** и **video**: http://html5doctor.com/video-canvas-magic/.

6 Микроданные и пользовательские данные

Кимберли Блессинг

6.0. Введение

Разметка, расширяющая контекст и увеличивающая значимость контента, — одно из основных преимуществ веб-стандартов, ставших результатом более чем 10-ти лет работы IT-евангелистов.

К середине 2000-х годов веб-стандарты стали довольно популярны, но «стандартизаторы» признавали, что разметке не хватает семантики для придания содержимому значимости. Например, на сайте хранятся персональные данные, а ссылки используются для описания отношений.

Для того чтобы решить эту проблему, разработчики сначала применяли формат XFN (XHTML Friends Network: http://gmpg.org/xfn/), а затем микроформаты (http://microformats.org). Оба решения основывались на атрибутах **class** и **rel**, валидных в (X)HTML. Однако эти атрибуты предназначены для других задач, поэтому некоторые гуру веб-разработки считают, что их использование для передачи дополнительной семантики — своего рода взлом.

Примерно в то же время рабочая группа W3C XHTML 2 создала спецификацию RDFa на основе Resource Description Framework, позволяющую повторно использовать существующие XHTML-атрибуты и добавлять новые для поддержки структурированных машиночитаемых данных.

Микроформаты быстро завоевали популярность у веб-профессионалов. Разработчики начали использовать их для разметки персональной информации,

резюме, кулинарных рецептов, а также для описания событий. Поисковики подхватили эту идею и стали обрабатывать дополнительные данные, полученные из микроформатов и RDFa, для улучшения результатов поиска.

С HTML5 веб-сообщество вернулось к строгому соблюдению спецификации. Теперь атрибуты **class** и **rel** в HTML5 (по крайней мере по спецификации WHATWG) используют *микроданные*. Это придает новому синтаксису дополнительную структурную семантику.

Кроме того, HTML5 предоставляет *атрибуты пользовательских данных*, которые в дальнейшем, в соответствии со стандартами, дадут специалистам возможность встраивать в разметку дополнительную информацию.

В современных браузерах подобная разметка уже работает, но еще не обеспечена связь для взаимодействия с микроданными DOM API или другими пользовательскими данными, широко поддерживаемыми пользовательскими агентами. Тем не менее, учитывая возможность создания сценария JavaScript для взаимодействия с дополнительными данными, нет оснований *не* пользоваться ими.

Как применять микроданные и пользовательские данные. Подобно микроформатам и RDFa, микроданные используются для разметки структурированной информации. Для этого вводятся новые атрибуты, применимые к любому элементу и определяемые в виде пары «имя/значение». Как и микроформаты, микроданные создают общие словари стандартизированной разметки данных, поэтому могут использоваться для обмена сведениями между сайтами и приложениями.

С атрибутами пользовательских данных все просто. Вы как программист можете создавать атрибуты, указывая префикс **data-**, и присваивать им значения. Пары «имя/значение» связаны с отображаемым контентом, но сами не выводятся. Поскольку атрибуты определяются в соответствии с потребностями сайта или приложения, они не применяются для обмена данными с другими сайтами и не учитываются поисковиками.

Следует ли использовать микроданные или RDFa для структурированной информации? Сейчас W3C уделяет большое внимание двум перекрывающим друг друга стандартам, что вызывает некоторое беспокойство о будущем этих спецификаций. Google, Microsoft и Yahoo!, тем временем, объединяются и содействуют развитию микроданных, создавая Schema.org с общей лексикой структур данных, доступных поисковым системам.

Ману Спорни (Manu Sporny) провел подробный анализ, сравнив микроданные, RDFa и микроформаты: http://manu.sporny.org/2011/uber-comparison-rdfa-md-uf/.

6.1. Добавление в разметку микроданных

Проблема

Необходимо добавить микроданные для придания контенту дополнительного смысла и обеспечения машинного доступа к информации.

Решение

Используйте атрибуты **itemscope** и **itemprop**, а при указании имен свойств обозначьте контент:

```
<p itemscope>
    <span itemprop="изобретатель">Тим Бернерс-Ли</span> создал
    <span itemprop="изобретение">Всемирную паутину</span>
</p>
```

Обсуждение

Атрибут **itemscope** применяется для определения области действия микроданных — *набора* пар «имя/значение». Значения атрибута **itemprop** устанавливают имена свойств и связанную с ними информацию, в данном случае содержимое тегов **span**. Таким образом, приведенный выше пример дает следующие пары «имя/значение»:

- Изобретатель: Тим Бернерс-Ли;
- Изобретение: Всемирная паутина.

Это очень простой пример. В следующем рецепте будет рассмотрен вариант реализации стандартизированного словаря.

Дополнительная информация

Спецификация HTML5-атрибута **itemprop**: http://www.w3.org/TR/html5/microdata.html#names:-the-itemprop-attribute.

6.2. Использование микроданных и Schema.org

Проблема

Необходимо придать контенту дополнительное значение, например он должен идентифицировать некую персону, чтобы популярные поисковые системы могли обработать эти сведения.

Решение

Кроме атрибутов **itemscope** и **itemprop**, укажите атрибут **itemtype** и используйте соответствующие имена свойств из лексики Schema.org:

```
<section itemscope itemtype="http://schema.org/Person">
    <h1 itemprop="name">Тим Бернерс-Ли</h1>
    <img itemprop="image" src="http://www.w3.org/Press/Stock/Berners-
        Lee/2001-europaeum-eighth.jpg">
    <p>
        <span itemprop="jobTitle">Руководитель</span>,
        <span itemprop="affiliation" itemscope
                itemtype="http://schema.org/Organization"
                itemprop="name">Консорциум Всемирной паутины</span>
    </p>
    <p itemprop="address" itemscope
        itemtype="http://schema.org/PostalAddress">
        <span itemprop="addressLocality">Кембридж</span>,
        <span itemprop="addressRegion">Массачусетс</span>
    </p>
    <a itemprop="url" href="http://www.w3.org/People/Berners- Lee/">
        Сайт W3C</a>
</section>
```

Обсуждение

Перед применением микроданных еще раз указывается элемент **section** с атрибутом **itemscope**, а к нему добавляется атрибут **itemtype**. В **itemtype** задается URL, определяющий тип элемента данных. В этом примере, чтобы идентифицировать данные, используется структура Schema.org.

Как и в предыдущем рецепте, атрибут **itemprop** применяется для определения имени свойства, придавая разметке дополнительное значение. Глядя на свойства и соответствующий им контент, можно сказать, что «Тим Бернерс-Ли» — это имя человека, а занимаемая им должность — «Руководитель».

Изображения и URL-свойства можно по-разному определять в **itemprop**: в этом примере они указаны как атрибуты **src** и **href** соответственно. Если вы уже работали с микроформатами, это не станет для вас сюрпризом.

В конце примера продемонстрировано, как указать в виде атрибута **itemprop** название организации и адрес. Эти данные вкладываются внутрь основного элемента. В обоих случаях **itemprop** не только определяет в теге свойства, непосредственно связанные с персоной, но и задает свойства атрибута **itemscope**. Пойдем дальше: **itemtype** применяется и для указания URL, описывающего тип данных элемента.

Хотя на первый взгляд это решение может показаться сложным, оно не сильно отличается от комбинирования нескольких микроформатов (например, hCard и hCalendar для резюме) или создания XML-объекта для представления вложенных данных.

Независимо от того, доводилось ли вам ранее работать над проектами, существует простой способ проверить, делаете ли вы успехи в применении словарей Schema.org. Вы можете использовать Google Rich Snippets Testing Tool (доступный по адресу http://www.google.com/webmasters/tools/richsnippets), чтобы убедиться, что ваша структурированная разметка данных будет корректно обработана (рис. 6.1).

Прежде чем Google, Microsoft и Yahoo! создали Schema.org, Google поддерживал Rich Snippets, основанный на лексике с сайта http://data-vocabulary.org. Документация Google для Rich Snippets до сих пор актуальна, но каждая страница ссылается на Schema.org.

Extracted rich snippet data from the page

Item
- **Type:** http://schema.org/person
- name = Tim Berners-Lee
- image = http://www.w3.org/Press/Stock/Berners-Lee/2001-europaeum-eighth.jpg
- jobtitle = Director
- affiliation = *Item*(1)
- address = *Item*(2)
- url = http://www.w3.org/People/Berners-Lee/

Item 1
- **Type:** http://schema.org/organization
- name = World Wide Web Consortium

Item 2
- **Type:** http://schema.org/postaladdress
- addresslocality = Cambridge
- addressregion = MA

Рис. 6.1. Результаты Google Rich Snippets Testing Tool

Дополнительная информация

Дополнительные типы данных, распознаваемые поисковыми системами, а также примеры кода можно найти на http://schema.org. Подробнее о микроданных читайте в статье от HTML5 Doctor (http://html5doctor.com/microdata/) и главе из «Погружения в HTML5» Марка Пилгрима (http://diveintohtml5.info/extensibility.html), где приводится подробное описание Google Rich Snippets.

6.3. Добавление в разметку пользовательских данных

Проблема

Необходимо присоединить к контенту дополнительные данные, не отображаемые для пользователя.

Решение

Определите собственные атрибуты **data-** для именования и хранения данных:

```
<h1>Мои модели Volkswagen</h1>
<ul>
    <li data-year="1996" data-color="белый"
        data-engine="VR6">Cabrio</li>
    <li data-year="1993" data-color="фиолетовый"
        data-engine="VR6">Corrado</li>
    <li data-year="2008" data-color="красный"
        data-engine="2.0T">Eos</li>
    <li data-year="2003" data-color="синий" data-engine="W8">Passat</li>
</ul>
```

Обсуждение

Поскольку не всех интересуют модели VW или просто автомобили, нет необходимости отображать такую информацию, как год выпуска, цвет и тип двигателя каждого VW. С помощью атрибутов пользовательских данных можно включать в разметку избыточные сведения, не видимые для пользователя.

Теперь такие данные поддерживаются и в HTML. Если хотите узнать, что еще можно с ними сделать, прочтите следующие рецепты этой главы.

Для того чтобы хранить сведения о годе выпуска, цвете и типе двигателя каждого автомобиля, в этом примере созданы три атрибута пользовательских данных: **data-year**, **data-color** и **data-engine**.

Как видите, все они начинаются с выражения **data-**, за которым следует хотя бы один символ. Это имя не может включать в себя прописные буквы, но может содержать дефисы. К примеру, разрешено определить атрибут **data-model-year**, но не допускается использовать атрибут **data-modelYear**.

Если атрибуты имеют в названии дополнительный дефис, то при использовании Dataset API необходимо перевести их в верблюжий регистр. Например, атрибут data-foo-bar должен называться dataset.fooBar. Но если вы используете getAttribute() или setAttribute(), то по-прежнему должны обращаться к нему как data-foo-bar.

Даже если последовательно применять три атрибута пользовательских данных из примера, это не даст специальных отношений, обеспечивающих взаимосвязь между атрибутами. Например, можно перечислить другие автомобили, применив только атрибут **data-color**, или перейти к списку приземлившихся самолетов, задействуя **data-engine**. Как программист, вы можете сохранить существующую структуру или самостоятельно определить пространство имен, необходимое для вашего сайта или приложения.

Дополнительная информация

Обсуждение атрибутов пользовательских данных от HTML5 Doctor: http://html5doctor.com/html5-custom-data-attributes/.

6.4. Доступ к пользовательским данным с помощью JavaScript

Проблема

Необходимо получить доступ к пользовательским данным на странице и выполнить на их основе некоторый алгоритм.

Решение

Начните с разметки, описанной в рецепте 6.3, и добавьте абзац **p** для вывода результатов сценария JavaScript:

```
<h1>Мои модели Volkswagen</h1>
<ul>
    <li data-year="1996" data-color="белый"
        data-engine="VR6">Cabrio</li>
    <li data-year="1993" data-color="фиолетовый"
        data-engine="VR6">Corrado</li>
    <li data-year="2008" data-color="красный"
        data-engine="2.0T">Eos</li>
    <li data-year="2003" data-color="синнй" data-engine="W8">Passat</li>
</ul>
<p></p>
```

Доступ к пользовательским данным с помощью Dataset API:

```
<script>
    var cars = document.getElementsByTagName("li");
    var output = "Какого цвета автомобили Кимберли? ";

    for (var i=0; i < cars.length; i++) {
        output += cars[i].dataset.color;
        if (i != (cars.length-1)) {
            output += ", "
        }
    }

    document.getElementsByTagName("p")[0].innerHTML = output;
</script>
```

Обсуждение

С введением атрибутов пользовательских данных для HTML5 также был определен Dataset DOM API — простой и удобный способ доступа к любой пользовательской инормации, связанной с любым элементом.

Сценарий JavaScript очень прост: создается массив из всех элементов списка (**cars**), а также формируется строка, которая вставляется в абзац **p (output)**. По мере перемещения по массиву **cars** с помощью **dataset.color** осуществляется доступ к каждому значению атрибута **data-color**, которое добавляется к переменной **output**. Конечным результатом станет фраза: «Какого цвета автомобили Кимберли? Белый, фиолетовый, красный, синий», которая добавляется в абзац в конце страницы.

Сейчас еще не все браузеры поддерживают Dataset API (табл. 6.1), но в них довольно просто обеспечить доступ к атрибутам пользовательских данных. Если Dataset не поддерживается, просто указывайте **getAttribute()**. Задавайте в цикле **for** операторы выбора и **getAttribute()**, если это необходимо:

```
for (i=0; i < cars.length; i++) {
    if (cars[i].dataset) {
        output += cars[i].dataset.color;
```

продолжение ⇩

```
        } else {
            output += cars[i].getAttribute("data-color");
        }
        if (i != (cars.length-1)) {
            output += ", "
        }
    }
```

Таблица 6.1. Поддержка Dataset API

IE	Firefox	Chrome	Safari	Opera	iOS	Android
–	6+	7+	5.1+	11.1+	–	–

Для того чтобы получать последние сведения о поддержке Dataset API, регулярно посещайте сайт http://caniuse.com/dataset.

Дополнительная информация

Спецификация HTML5-атрибута **itemprop**: http://www.w3.org/TR/html5/microdata.html#names:-the-itemprop-attribute.

6.5. Управление пользовательскими данными

Проблема

Необходимо манипулировать существующими на странице пользовательскими данными или добавлять новые данные.

Решение

Воспользуйтесь кодом из рецепта 6.4. Для начала используйте ту же разметку:

```
<h1>Мои модели Volkswagen</h1>
<ul>
    <li data-year="1996" data-color="белый"
        data-engine="VR6">Cabrio</li>
    <li data-year="1993" data-color="фиолетовый"
        data-engine="VR6">Corrado</li>
```

```
    <li data-year="2008" data-color="красный"
        data-engine="2.0T">Eos</li>
    <li data-year="2003" data-color="белый" data-engine="W8">Passat</li>
</ul>
<p></p>
```

и с помощью Dataset API изменяйте и добавляйте собственные данные, а также **setAttribute()** для обеспечения обратной связи:

```
<script>
    var cars = document.getElementsByTagName("li");
    for (var i=0; i < cars.length; i++) {
        if (cars[i].dataset) {
            cars[i].dataset.color = "желтый";
            cars[i].dataset.rating = "удивительный";
        } else {
            cars[i].setAttribute("data-color", "желтый");
            cars[i].setAttribute("data-rating", "удивительный");
        }
    }

    var output = "Какого цвета автомобили Кимберли? ";

    for (var i=0; i < cars.length; i++) {
        if (cars[i].dataset) {
            output += cars[i].dataset.color;
        } else {
            output += cars[i].getAttribute("data-color");
        }
        if (i != (cars.length-1)) {
            output += ", "
        }
    }

    document.getElementsByTagName("p")[0].innerHTML = output;
</script>
```

Обсуждение

Если вы знаете, как получить доступ к пользовательским данным, то манипулировать ими довольно легко. Если вы можете применять Dataset API, просто присвойте новое значение нужному атрибуту, а в противном случае воспользуйтесь **setAttribute()**. Этот метод также позволяет добавлять новые атрибуты пользовательских данных.

В этом примере решено с помощью JavaScript перекрасить все автомобили в желтый цвет. Проходя по списку элементов, можно получить доступ к **dataset.color** (или, используя **setAttribute()**, к **data-color**) и задать новое значение **"желтый"**.

Таким же образом используйте JavaScript, чтобы добавить для каждого автомобиля атрибут **data-rating**. В том же цикле создайте **data-rating** со значением **"удивительный"** с помощью **dataset.rating** (или используя для этих целей **setAttribute()**).

Если необходимо удалить атрибут пользовательских данных, можно установить его значение равным **null**.

С помощью Opera Dragonfly убедитесь, что изменения успешно внесены в DOM (рис. 6.2).

Дополнительная информация

Спецификация Dataset: http://dev.w3.org/html5/spec/Overview.html#dom-dataset.

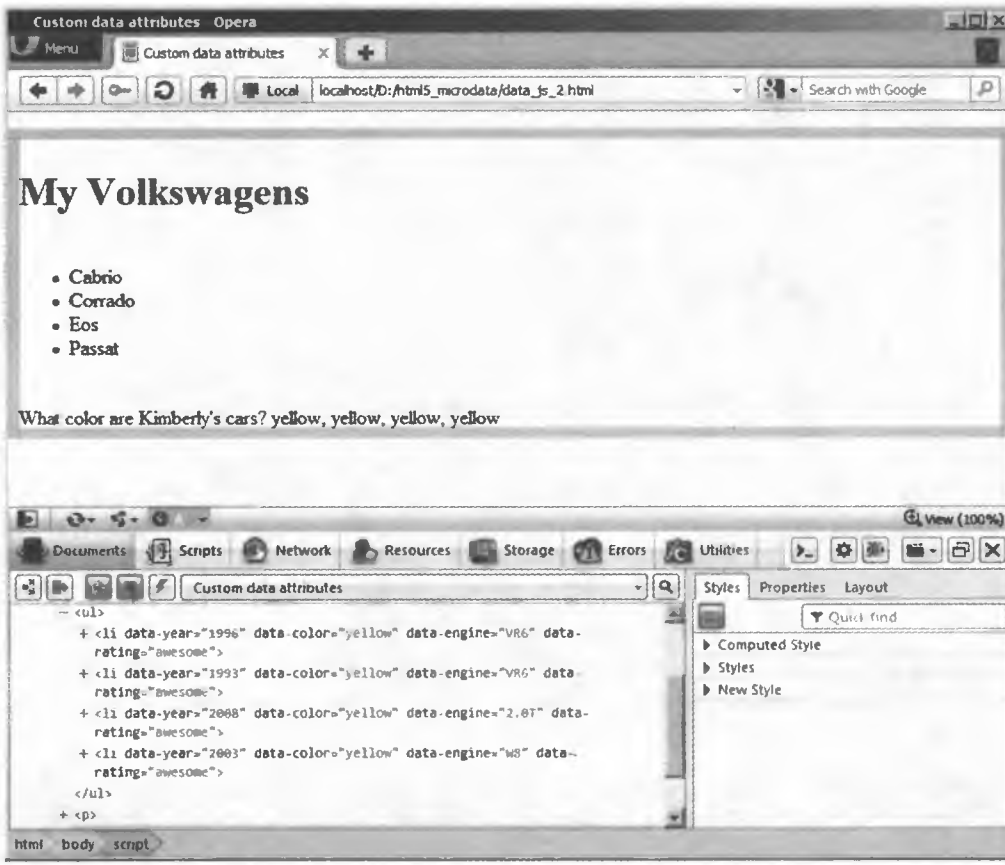

Рис. 6.2. Просмотр изменений Dataset с помощью Dragonfly в Opera 11.1

6.6. Создание приложения для карты с помощью пользовательских данных

Проблема

Необходимо создать метки на карте с помощью атрибутов пользовательских данных.

Решение

Сохраните широту и долготу места, которое необходимо показать на карте, в специальных атрибутах пользовательских данных. Затем напишите сценарий JavaScript, обеспечивающий доступ к ним и создающий метки на карте:

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример карты</title>
    <script type="text/javascript"
            src="http://maps.google.com/maps/api/js?sensor=false">
    </script>
    <style type="text/css">
        #map { height:500px; width:500px; border:1px solid #000; }
    </style>
</head>
<body>
<h1>Мороженое в Филадельфии</h1>
<ul>
    <li><a    data-lat="39.9530255" data-long="-75.1596066"
            href="http://www.bassettsicecream.com/">Bassetts Ice
            Cream</a></li>
    <li><a    data-lat="39.949888" data-long="-75.161717"
            href="http://www.capogirogelato.com/">Capogiro
            Gelateria</a></li>
    <li><a    data-lat="39.949556" data-long="-75.1428795"
            href="http://www.franklinfountain.com/">Franklin
            Fountain</a></li>
</ul>

<div id="map"></div>

<script>
    var map_options = {
        zoom: 15,
        center: new google.maps.LatLng(39.95, -75.152),
        mapTypeId: google.maps.MapTypeId.HYBRID
    };

    var map = new google.maps.Map(document.getElementById("map"),
            map_options);
```

продолжение

```
    var locations = document.getElementsByTagName("a");
    for (var i=0; i < locations.length; i++) {
        var latitude, longitude;
        if (locations[i].dataset) {
            latitude = locations[i].dataset.lat;
            longitude = locations[i].dataset.long;
        } else {
            latitude = locations[i].getAttribute("data-lat");
            longitude = locations[i].getAttribute("data-long");
        }
        locations[i][i] = new google.maps.Marker({
            position: new google.maps.LatLng(latitude, longitude),
            title: locations[i].innerHTML,
            map: map
        });
    }
</script>
</body>
</html>
```

Обсуждение

Вам уже известно, как определить пользовательские данные, поэтому создать список меток с атрибутами **data-latitude** и **data-longitude** не составит труда.

Для создания Google-карты с помощью Google API достаточно вызвать API в разделе **head** документа, указать в **body** элемент для размещения карты, а затем вызвать сценарий, определяющий настройки карты и отображающий ее.

После того как сценарий создаст изображение карты, он получит доступ к данным о широте и долготе объектов и использует их для меток на карте.

Пример: JQuery метод data(). Начиная с версии 1.4.3, библиотека JQuery обеспечивает доступ к пользовательским данным и управление ими. В JQuery 1.6 уже внесены изменения, учитывающие спецификацию HTML5, поэтому по возможности используйте именно эту версию (http://www.webmonkey.com/2011/05/jquery-update-improves-html5-data-tools/).

JQuery-метод **data()** довольно прост. Чтобы увидеть, как он работает, вернемся к предыдущему примеру:

```
<ul>
    <li data-year="1996" data-color="white"
        data-engine="VR6">Cabrio</li>
</ul>

<script>
    alert($("li").data("year"));   // выводит "1996"
    alert($("li").data("engine")); // выводит "VR6"
```

```
    // снова перекрасим автомобиль
    $("li").data("color", "желтый");

    // и добавим данные о рейтинге
    $("li").data("rating", "удивительный");
</script>
```

Метод **data()** успешно считывает атрибуты **data-year** и **data-engine**, определенные в разметке. Однако если вы получаете доступ к **data-color** с помощью Dataset API или просматриваете объект DOM, вы не увидите, как «пожелтеет» его значение, и не найдете атрибут **data-rating**, как это было сделано в рецепте 6.5:

```
<script>
    alert($("li").data("color")); // выводит "желтый"
    alert(document.getElementsByTagName("li")[0].dataset.color);
    // выводит "белый"
</script>
```

JQuery, считывая информацию из атрибутов пользовательских данных, не записывает ее обратно в DOM, а хранит в объекте JavaScript. Это ускоряет работу приложений для доступа или манипуляции множеством данных. Однако если вы забудете о такой особенности и попытаетесь использовать в своих приложениях Dataset API, то столкнетесь с неожиданными результатами.

Дополнительная информация

Несколько приложений, использующих микроданные:

- PaintbrushJS Дэйва Ши (Dave Shea): http://mezzoblue.github.com/PaintbrushJS/demo/inde7.html.
- Dynamic Google Analytics Tagging Джейсона Карнса: http://jasonkarns.com/blog/2010/03/10/google-analytics-tagging/.
- Сайт SXSW 2010 Кристофера Шмитта, Кайла Симпсона, Стефани Салливан и Зои Гилленвотер (Zoe Gillenwater): http://www.sxswcss3.com.

7 Доступность

Анита Павка

7.0. Введение

«Инвалидность» — довольно широкое понятие. Основные категории физических недостатков, влияющих на пользование Сетью, включают проблемы со зрением, слухом, восприятием, речью (особенно актуально для мобильных устройств) и ограниченными двигательными возможностями. Одна и та же категория физических недостатков может проявляться в различных формах. Например, проблемы со зрением включают дальтонизм, но не ограничиваются этим.

От 7 до 10 % мужчин не способны различать красный и зеленый цвета.

Физические недостатки могут иметь разные степени, и один и тот же человек может иметь несколько проблем. Запястья бывают сломаны или ампутированы, в зависимости от этого потеря трудоспособности считается временной или постоянной. Постоянная инвалидность может оказаться врожденной или приобретенной.

Таким образом, любой человек может стать инвалидом.

Цель — обеспечение доступности

Веб-доступность позволяет устранить барьеры, мешающие людям использовать Интернет. Можно сказать, что доступность лежит в основе Сети, цель которой — сломать барьеры общения.

По словам Консорциума Всемирной паутины (http://www.w3.org/standards/webdesign/accessibility):

> Сеть принципиально предназначена для работы всех людей, независимо от аппаратного и программного обеспечения, языка, культуры, местоположения, физических или умственных способностей. Когда Сеть отвечает этой цели, она доступна людям с различными слуховыми, двигательными, зрительными и когнитивными способностями.

Веб-программирование и разметка — важные этапы на пути устранения барьеров. Но если что-то сделано неправильно, могут возникнуть новые проблемы.

Аппаратное и программное обеспечение, используемое для доступа к веб-контенту, — не менее важный инструмент. Пользовательский агент, например браузер, и любые вспомогательные технологии, как программы чтения с экрана, вслух читающие веб-контент, должны суметь правильно интерпретировать и представить пользователям информацию или запрограммированную вами функциональность.

Принципы доступности

К счастью, существует несколько принципов с рекомендациями для веб-дизайнеров и разработчиков. Если вы работаете с правительственными или финансируемыми государством сайтами, то можете следовать местным или федеральным законодательным актам.

Кроме того, вы, возможно, слышали про W3C Web Accessibility Initiative (WAI) (Инициатива доступности Сети) или W3C Web Content Accessibility Guidelines (WCAG) (Руководство по обеспечению доступности Сети). W3C WAI — это обновление оригинального WCAG. Через девять лет после выпуска первой версии в декабре 2008 года WCAG 2.0 стал рекомендацией W3C. Эти руководящие принципы включают некоторые международные рекомендации, позволяющие сделать веб-контент более доступным и полезным для людей с ограниченными возможностями.

WCAG 2.0 использует аббревиатуру POUR, обобщающую четыре принципа доступности сайтов. POUR основывается на Perceivable (Воспринимаемости), Operable (Работоспособности), Understandable (Понятности) и Robust (Надежности).

- *Воспринимаемость* — означает, что информация должна быть доступной для восприятия, как правило, зрительно или на слух. Необходимо предоставить альтернативный текст для изображений, заголовки для аудио и видео, а также обеспечить достаточный контраст между текстом и фоном.

- *Работоспособность* — гарантирует, что посетители смогут взаимодействовать между собой и пользоваться сайтом через навигацию, контент, формы и динамический контроль.
- *Понятность* — означает, что текст должен быть четким, а поведение сайта — последовательным и предсказуемым.
- *Надежность* — определяет разметку страницы и программирует работу с различными пользовательскими агентами и вспомогательными технологиями.

Кроме W3C WAI, был создан WAI-ARIA 1.0 (Accessible Rich Internet Applications (Стандарт доступности активных интернет-приложений)). Он позволяет добавлять семантическое значение для Ajax-подобного, динамического веб-контента и пользовательских виджетов путем определения набора ролей и состояний, а также свойств этих ролей.

Другими словами, ARIA дает возможность программно взаимодействовать с вспомогательными технологиями, которые показывают изменения на странице в ответ на действия пользователей, а не только то, что присутствует в оригинальной разметке. Это становится все более важно: в докладе о распространении технологии SecuritySpace говорится, что с января 2010 года более 64 % сайтов содержат JavaScript (http://www.securityspace.com/s_survey/data/man.200912/techpen.html).

Работа над спецификацией ARIA до сих пор не окончена. В то время как она постоянно совершенствуется, иногда встречаются ошибки и отмечается недостаточно полная поддержка пользовательскими агентами.

Хорошо то, что *реализовывать* ARIA нетрудно. Это точно не сложнее, чем создавать кроссбраузерные, пользовательские JavaScript-виджеты. Вы ведь уже это умеете, верно?

7.1. Создание текстового описания рисунка

Проблема

Необходимо добавить для изображения альтернативный текст.

Решение

Добавьте альтернативный программный текст с помощью атрибута **alt** изображения.

Если изображение может быть заменено альтернативным текстом, не превышающим одного-двух предложений, добавьте этот текст в качестве значения атрибута **alt**:

```
<img src="next_button.jpg" alt="Переход к следующей странице.">
```

Альтернативный текст не должен превышать 150 символов.

Известно ли вам, насколько хорошо работает подобный текст? Замечательное правило: «Если вы замените каждое изображение на странице альтернативным текстом, то страница будет передавать тот же смысл, что и при наличии изображений».

Если это сложное изображение, например график или диаграмма, и его нельзя просто обобщить, включив дополнительную информацию или ссылку на картинку, то необходим более подробный и длинный альтернативный текст:

```
<a href="diagram_desc.html"><img src="diagram.jpg" alt="Схема,
    Демонстрирующая, как собрать изделие, с ссылкой на инструкцию
    по монтажу"></a>
```

Если вы предоставляете дополнительную информацию не только в атрибуте alt, убедитесь, что текст обновляется каждый раз, когда вносятся изменения, связанные с изображением.

Несколько изображений. Если для обозначения изображения необходимо использовать текст из нескольких альтернативных источников, и весь этот контент находится на той же веб-странице, используйте атрибуты ARIA **aria-labelledby** или **aria-describedby**. Они ссылаются на альтернативный текст и могут включать в себя несколько ссылок.

В этом примере атрибут **aria-describedby** в элементе **img** ссылается на значения атрибутов **id**, используемых при описании элементов **h1** и **p**:

```
<h1 id="johnny1">Джонни бросает торт в свою маму, пока папа отвернулся
    от нее</h1>
<p id="johnny2">Бедная мама Джонни размазывает торт. Ну вот, рубашка
    испорчена!</p>
...
<img src="johnny_1st_bd.jpg" aria-describedby="johnny1 johnny2"
    alt="Первый день Рождения Джонни">
```

Пока пользовательские агенты и вспомогательные технологии лучше поддерживают элементы **figure** и **figcaption**, используйте атрибут **aria-labelledby** для обеспечения связи между изображением в элементе **figure** и его подписью.

Добавьте атрибут **aria-labelledby** в элемент **img** и свяжите с ним значение для атрибута **id** элемента **figcaption**:

```
<figure>
    <img src="ceremony_photo.jpg" aria-labelledby="figcaption123">
    <figcaption id="figcaption123">
        Торжественное открытие нового здания библиотеки государственного
        университета.
    </figcaption>
</figure>
```

Обсуждение

Альтернативный текст для нетекстовых элементов, например изображений, необязательно должен быть *описанием*. Напротив, он должен служить той же цели и давать ту же информацию, что и нетекстовый элемент, и выступать в качестве его замены.

Предоставление альтернативного текста для нетекстовых элементов работает в соответствии с WCAG 2.0. Тем не менее на момент написания статьи атрибут **alt** технически не описан в HTML5. Однако рекомендуется по возможности использовать его для изображений. Любое решение о его удалении, задании нулевого значения либо использовании другого метода, обеспечивающего альтернативный текст, должно быть осознано и тщательно рассмотрено.

Альтернативный текст для изображений должен:

- включать весь текст, который появляется на изображении;
- содержать контекст изображения, определяющий, что в нем важно;
- если это необходимо, передавать цель или функцию изображения;
- исключать информацию, доступную в виде текста рядом с изображением;
- не содержать таких фраз, как «образ» или «картины»;
- быть кратким.

Нулевой атрибут alt

Нулевой атрибут **alt** — это **alt**, значение которого не содержит пробелов или текста. Это не значит, что можно вообще пропустить его:

```
<img src="border_decoration.jpg" alt="">
```

Добавление нулевого или пустого атрибута **alt** уместно в следующих ситуациях.

- Изображение представляет собой декоративный элемент.

- Альтернативный текст доступен в непосредственной близости от изображения. Тем не менее этот совет некорректен, если изображение является ссылкой.
- Изображение входит в группу рисунков, формирующих общую картину, и такое составное изображение уже имеет альтернативный текст.
- Изображение не должно быть видно пользователю (например, рисунок размером 1 пиксел, показывающий количество просмотров страницы).

Альтернативный текст также не требуется, если вы вместо элемента img используете CSS-свойство background-image, добавляя на веб-страницу исключительно декоративные изображения.

Следующее CSS-правило связывает изображение с пользовательским классом. Само правило может относиться с HTML-элементу, например div или body:

```
.border {
    background-image: url(border_decoration.jpg);
    background-repeat: repeat-x;
}
```

Поскольку вспомогательные технологии могут полностью игнорировать изображения с нулевым атрибутом **alt**, то пользователи не будут знать о существовании таких необходимых вам изображений, ведь они могут оказаться декоративными без взаимодействия с веб-страницей.

Однако если вы вообще опустите атрибут **alt** в элементе **img**, то вспомогательные технологии сообщат, что изображение существует на странице, и сделают все возможное, чтобы предоставить информацию о нем, даже если это будет просто имя файла. При рассмотрении вопроса об использовании нулевого атрибута **alt** или другого атрибута имейте в виду, что существуют различия в обработке некоторых вспомогательных технологий.

Учтите, что атрибут longdesc (http://www.w3.org/TR/REC-html40/struct/objects.html#adef-longdesc-IMG), вероятно, будет исключен из HTML5. Вместо него будет использоваться либо ссылка на страницу с более подробным альтернативным текстом, либо информация, прилагаемая к изображению, как было ранее описано в этом рецепте.

Использование figcaption

В HTML5 изображение может быть заключено в элементах **figure** и **figcaption**, которые либо предоставляют весь альтернативный текст, либо дополняют текст, предусмотренный атрибутом изображения **alt**.

На практике атрибут alt поддерживает более универсальный метод для предоставления информации об изображении. Это будет происходить до тех пор, пока элемент figcaption продолжит в достаточной мере поддерживаться браузерами и вспомогательными технологиями.

Как упоминалось ранее, некоторые вспомогательные технологии полностью игнорируют изображения с пустым атрибутом **alt**. Чтобы спрятать рисунок, не добавляйте атрибут **alt**, если не планируете устанавливать его значение:

```
<figure>
    <img src="ceremony_photo.jpg">
    <figcaption>
        Торжественное открытие нового здания библиотеки государственного
        университета.
    </figcaption>
</figure>
```

Дополнительная информация

«HTML5: Методы предоставления полезного альтернативного текста»: http://dev.w3.org/html5/alt-techniques/.

7.2. Определение аббревиатур и сокращений

Проблема

Необходимо обозначить на странице аббревиатуры и сокращения.

Решение

Добавьте в состав страницы *полную* расшифровку сокращения.

Как правило, аббревиатуры помещаются в скобках сразу после их полной расшифровки. Включите этот параметр, когда сокращение появляется на странице в первый раз:

```
Web Content Accessibility Guidelines (WCAG) 2.0 был выпущен как
рекомендация World Wide Web Consortium (W3C) в декабре 2008 года.
```

Использование abbr. Введите элемент **abbr** для программного связывания полной расшифровки с сокращением. Это встроенный элемент, который применяется в блочных или других встроенных элементах:

```
<p><abbr title="Web Content Accessibility Guidelines">WCAG</abbr> 2.0
был выпущен как рекомендация <abbr title="World Wide Web Consortium">
W3C</abbr> в декабре 2008 года.</p>
```

Если сокращение употребляется во множественном числе или притяжательном наклонении, убедитесь, что содержимое элемента **abbr** соответствует тому, что входит в атрибут **title** (рис. 7.1):

```
<p><abbr title="Миссия Национального комитета по астронавтике
и исследованию космического пространства "> впоследствии планирует
запуск NASA</abbr> <abbr title="Космическая транспортная
система">STS</abbr>-133.<p>
```

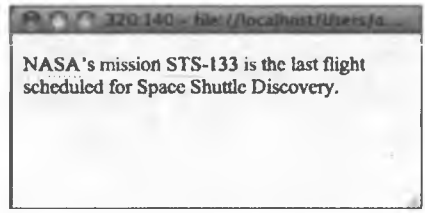

Рис. 7.1. Аббревиатуры с исопользованием подчеркивания

Обсуждение

Как отмечается в рецепте 1.13, элемент **acronym** в HTML5 считается устаревшим. Это исключает трудности с XHTML по целесообразности использования элементов **abbr** или **acronym**: элемент **abbr**, по сути, объединяет в себе оба варианта.

Стили abbr. Рекомендуется визуально стилизовать элемент **abbr** для зрячих пользователей, для которых доступен дополнительный контент. По умолчанию стиль для **abbr** в браузерах Opera и Firefox отображается с подчеркиванием (нижняя пунктирная граница) под текстом. Можно одной CSS-строкой добавить этот эффект в Safari, Chrome и Internet Explorer 8:

```
abbr { border-bottom: 1px dotted black;}
```

Когда пользователь наводит указатель мыши на аббревиатуру, большинство браузеров показывают полную расшифровку слова или фразы.

Аббревиатура — это сокращенная форма слова или фразы, например CSS. Сокращение — один из видов аббревиатуры: это новое слово, которое образуется из начальных букв нескольких слов. Вы произносите эти буквы как одно слово, например «НАСА» и «НАТО».

Дополнительная информация

Если хотите узнать, как обойти IE версии 8 и ниже, читайте блог WHATWG «Поддержка новых элементов в IE»: http://blog.whatwg.org/supporting-new-elements-in-ie.

7.3. Определение разделов страницы с помощью ключевых ролей ARIA

Проблема

Необходимо выделить общие разделы контента страницы.

Решение

Добавьте «ориентиры» ARIA, повышающие качество распознавания HTML5-элементов вспомогательными технологиями.

До сих пор большинство вспомогательных технологий не поддерживает HTML5 (на момент выхода в печать этой книги), и, чтобы придать значимость новым HTML-элементам, вы можете добавить ключевые роли ARIA (рис. 7.2). Ниже перечислены связанные между собой роли ARIA и HTML5-элементы. Это *не* жесткие правила, а скорее полезные советы.

Роли ARIA

Добавьте в раздел **header** страницы роль **banner** (обычно содержащую логотип/название сайта и другую описательную информацию). Используйте ее для каждого документа или веб-приложения только один раз:

```
<header role="banner">
```

Добавьте роль **complementary** для элемента **aside**. Они предназначены для разметки текста, связанного с основным контентом. Не используйте эту роль или элемент для совершенно разной и ничем не объединенной информации:

```
<aside role="complementary">
```

Введите роль **navigation** для каждого элемента **nav**:

```
<nav role="navigation">
```

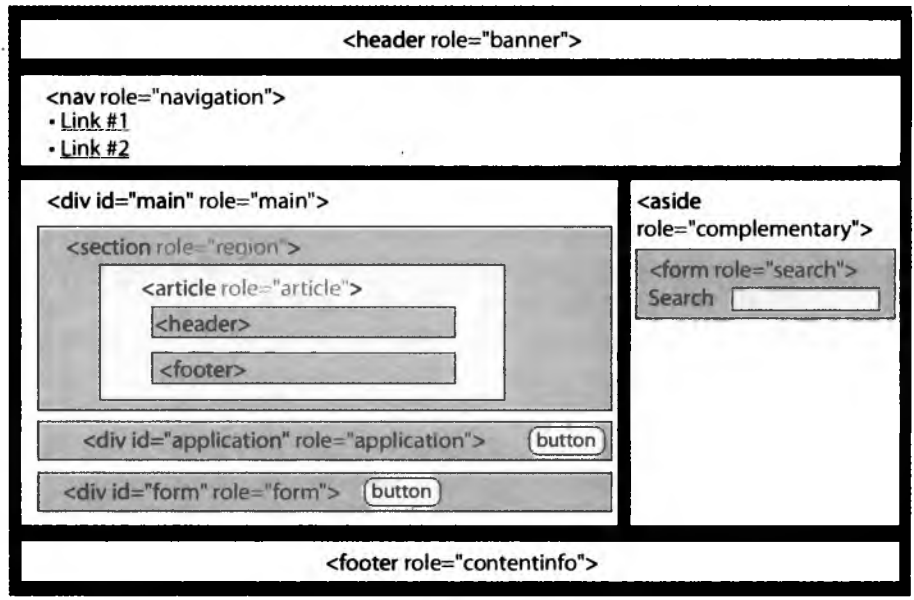

Рис. 7.2. Демонстрация использования ролей ARIA в HTML5-документе

Добавьте роль **form** для любого элемента **form**, если форма позволяет осуществлять поиск:

```
<form role="form">
```

Добавьте роль **search** для формы поиска:

```
<form role="search">
```

Роль **contentinfo** определяет информацию о таком контенте страницы, как авторские права и условия конфиденциальности. Используйте **contentinfo** только один раз для каждого документа или веб-приложения. Если в подвале представлен только этот тип информации, свяжите роль с элементом **footer**:

```
<footer role="contentinfo">
    <p>Авторские права 2011</p>
</footer>
```

Если у вас «толстый подвал» с большим количеством другой информации или ссылок, поместите материал **contentinfo** в элемент-контейнер (**div** или **p**) и назначьте для контейнера роль **contentinfo**:

```
<footer>
...
    <p role="contentInfo">Авторские права 2011</p>
...
</footer>
```

Роли для веб-приложений

Некоторые ключевые роли ARIA, например **application** и **main**, являются уникальными и прямо не соответствуют конкретным HTML5-элементам.

Добавьте HTML-элементу, содержащему веб-приложение вместо обычного текста, роль **application**. Убедитесь, что HTML-элемент, связанный с этой ролью, включает в себя все веб-приложение. Роль **application** может давать вспомогательным технологиям инструкции для перехода в режим, который лучше подходит для работы с приложением. После приложения, на этой же странице, где продолжается обычный текст, вы можете добавить роль **document**:

```
<div role="application">
...
</div>
```

Добавьте роль **main** для HTML-элементов, содержащих основную информацию документа, например **article**, **div** и т. д.

```
<div role="main">
```

В будущем это решение может стать альтернативой «перехода к основному контенту» или ссылкам для «пропуска навигации». Другими словами, эта роль может предоставить пользователям вспомогательных технологий больше возможностей для перехода непосредственно к основному контенту страницы вместо блуждания по длинным спискам ссылок и текста, повторяющимся на каждой странице сайта.

Если статья оказывается главным контентом на странице, добавьте элементу **article** роль **main**:

```
<article role="main">
```

Обсуждение

ARIA расширяет доступность HTML. Вспомогательные технологии могут использовать ключевые роли ARIA для определения контента и перехода по его разделам. Эти функции обеспечивают более надежную и последовательную работу пользователей.

Вы можете задать на странице несколько элементов **nav**, но в них должны быть только основные блоки навигационных ссылок. Элемент **footer** — пример ссылки, которая не нуждается в элементе **nav**. Конечно, если вы создаете «толстый подвал», как на сайте W3C (http://www.w3.org), будет целесообразно использовать в нем элемент **nav**.

Более подробную информацию об оформлении «толстого подвала» можно найти на сайте UI-шаблонов http://ui-patterns.com/patterns/FatFooter.

Дополнительная информация

Обзор ошибок, возникающих в программах для чтения с экрана и связанных с HTML5-элементами в сочетании с ключевыми ролями ARIA, приведен на странице http://www.accessibleculture.org/research/html5-aria, а обходные пути ошибок некоторых вспомогательных технологий вы найдете по адресу http://www.accessibleculture.org/blog/2010/11/html5-plus-aria-sanity-check/.

7.4. Создание доступных навигационных ссылок

Проблема

Необходимо обозначить навигационные ссылки, сделав их удобными большинству пользователей.

Решение

Поместите список навигационных ссылок внутри HTML5-элемента **nav**. Задайте неупорядоченный список ссылок, если последовательность их посещения не важна:

```
<nav role="navigation">
    <ul>
        <li><a href="#">Главная</a></li>
        <li><a href="#">О проекте</a></li>
        <li><a href="#">Блог</a></li>
        <li><a href="#">Портфолио</a></li>
    </ul>
</nav>
```

В ином случае используйте для разметки ссылок упорядоченный список:

```
<nav role="navigation">
    <ol>
        <li><a href="#">Глава 1</a></li>
        <li><a href="#"> Глава 2</a></li>
        <li><a href="#"> Глава 3</a></li>
    </ol>
</nav>
```

Затем добавьте ссылку для пропуска навигации:

```
<div id="offscreen"><a href="#maincontent">Перейти к основному
    контенту</a></div>
...
<nav role="navigation">
    <ul>
        <li><a href="#">Главная</a></li>
        <li><a href="#">О проекте</a></li>
        <li><a href="#">Блог</a></li>
        <li><a href="#">Портфолио</a></li>
    </ul>
    </nav>
    ...
    <article id="maincontent" role="main">
```

Эта ссылка может находиться перед разделом навигации или ближе к верхней части страницы, в зависимости от того, насколько контент важнее навигации. Ее цель — предоставление клавиатуры и вспомогательных технологий, позволяющих перейти непосредственно к основному контенту страницы.

Обсуждение

Список, независимо от типа, как правило, считается наиболее подходящим HTML-элементом для групп навигационных ссылок. Все началось с тех пор, как Кристофер Шмитт впервые использовал список в этих целях (http://www.alistapart.com/articles/practicalcss/).

Подобная разметка может сообщить пользователю вспомогательных технологий, сколько навигационных элементов перечислено в списке и в какой части списка в данный момент находится пользователь.

Возможно, в будущем вам не придется добавлять роли ARIA или ссылки для пропуска навигации. Пользовательские агенты и вспомогательные технологии должны распознавать каждый элемент раздела и позволять посетителям перемещаться между разделами страницы. Скорее всего, вам нужно будет только это:

```
<nav role="navigation">
    <ul>
        <li><a href="#">На главную</a></li>
        <li><a href="#">О проекте</a></li>
        <li><a href="#">Блог</a></li>
        <li><a href="#">Портфолио</a></li>
    </ul>
</nav>
...
<article>
```

Использование ссылок для пропуска навигации

Ссылки для пропуска навигации позволяют пользователям миновать контент и ссылки, размещенные на каждой странице сайта, что очень важно на сайтах с длинными списками ссылок, например сайтах для электронной торговли.

Самый простой способ реализовать ссылку пропуска навигации — добавить в качестве якоря текстовую ссылку. Ее точное местоположение на экране не имеет значения, поскольку ссылка будет первым элементом при переходе на страницу, и ссылка-якорь окажется там, где начинается основной контент страницы.

Скрытие ссылок для «пропуска навигации» для основного контента

Если вы хотите «спрятать» ссылки пропуска навигации от зрячих пользователей, используйте CSS. Но прежде убедитесь, что это не смутит пользователей, работающих с клавиатурой, ведь текст ссылки не будет виден, но пользователь сможет перейти по ней:

```
.offscreen {
    left: -9999em;
    position: absolute;
    top: auto;
    overflow: hidden;
    width: 1px;
    height: 1px;
}
```

Старайтесь не использовать display:none, чтобы скрыть пропуск навигационных ссылок, так как программы чтения с экрана могут полностью проигнорировать ссылки.

Вместо того чтобы скрывать такие ссылки, рассмотрите возможность использования CSS для задания их стиля, соответствующего вашему дизайну. Или, что еще лучше, постарайтесь, чтобы такие ссылки становились видимыми только при фокусировке с клавиатуры (http://webaim.org/temp/skipcss3.htm).

Дополнительная информация

Рецепт 7.3 и статья по использованию «пропуска навигационных» ссылок: http://webaim.org/techniques/skipnav/.

7.5. Связывание полей формы с метками

Проблема

Необходимо определить текст метки для каждого поля формы.

Решение

Для того чтобы программно связать текст с полем формы, используйте элемент **label**. Значение атрибута **for** метки обязано совпадать со значением атрибута **id** поля формы. Значение **id** должно быть уникальным для каждой страницы:

```
<label for="fName">Имя</label>
<input type="text" id="fname">
```

Не следует использовать элементы **label** с полями типа **hidden**.

Несколько меток. Добавьте ARIA-атрибуты **labeledby** или **describedby**, если нужно поместить на страницу несколько меток или описаний, чтобы передать смысл поля формы. Оба атрибута могут содержать ссылки на несколько значений. Разделяйте значения пробелами:

```
<form>
    <fieldset>
        <legend>Выход из учетной записи</legend>
        <span id="labelAutoLogout">Автоматически выйти через</span>
        <input id="autoLogout" type="text" value="30"
               aria-labelledby="labelAutoLogout labelAutoLogoutTime"
               aria-describedby="autoLogoutDesc">
        <span id="labelAutoLogoutTime">минут неактивности</span>
        <p id="autoLogoutDesc">Позволяет настраивать время ожидания для
            каждой вашей учетной записи.</p>
    </fieldset>
</form>
```

Обсуждение

Элемент **label** предназначен для описания соответствующих полей формы. Он лучше всего подходит для простых форм, в то время как ARIA выгоднее применять для полей с несколькими метками или в тех случаях, когда элемент **label** недоступен.

Имейте в виду, что основное различие между **aria-labelledby** и **aria-describedby** состоит в том, что первый предназначен для *краткого* описания, в то время как второй может содержать *много слов*.

Дополнительная информация

HTML5-спецификация **label** находится по адресу http://dev.w3.org/html5/markup/label.html.

7.6. Логическая группировка полей формы

Проблема

Необходимо сгруппировать области, относящиеся к форме.

Решение

Можно добавить элемент **fieldset** вокруг каждой группы полей формы, а также установить элемент **legend** в качестве первого элемента в **fieldset**. Он представляет собой заголовок или название для группы полей:

```
<form>
    <fieldset>
    <legend>Рейтинг фильмов</legend>
    <p><input type="radio" name="rating" id="rating1">
    <label for="rating1">Отлично</label></p>
    <p><input type="radio" name="rating" id="rating2">
    <label for="rating2">Хорошо</label></p>
    <p><input type="radio" name="rating" id="rating3">
    <label for="rating3">Неплохо</label></p>
    <p><input type="radio" name="rating" id="rating4">
    <label for="rating4">Плохо</label></p>
    </fieldset>
</form>
```

Делайте все возможное, чтобы текст **legend** был кратким.

Для некоторых вспомогательных технологий элемент legend объявляется до метки каждого поля в элементе fieldset. Это может быть довольно утомительно.

Обсуждение

Элемент **fieldset** позволяет организовать поля формы и помогает пользователям понять ее цель. Он также может применяться для групп переключателей или флажков.

Элементы **legend** и **fieldset** могут быть оформлены с помощью CSS. По умолчанию большинство браузеров отображает сплошную границу шириной 1 пиксел вокруг **fieldset** и в верхнем левом углу контейнера появляется **legend** (рис. 7.3).

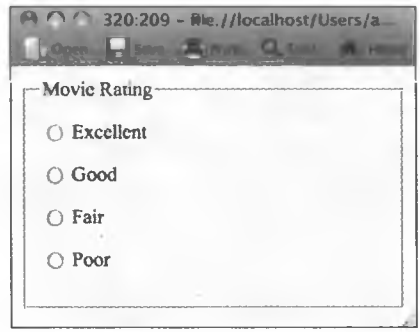

Рис. 7.3. Отображение элементов fieldset и legend по умолчанию в Opera 11

Группировка полей формы с помощью элементов **fieldset** очень полезна, если последовательность полей важна для пользователей клавиатур. Если вы изменяете значение свойства **tabindex**, то порядок обхода формы обычно соответствует порядку появления полей в исходном коде.

Порядок следования полей было сложно определить при использовании таблиц в качестве макета формы. Разметка средствами CSS способна сгладить эту проблему, поскольку поля в исходном коде могут быть перечислены в логическом порядке, а затем позиционироваться на экране независимо от места в коде.

Дополнительная информация

Статья «Создание фокусируемых элементов с помощью tabindex» на странице http://snook.ca/archives/accessibility_and_usability/elements_focusable_with_tabindex и HTML5-спецификация элемента **fieldset** по адресу http://dev.w3.org/html5/markup/fieldset.html.

7.7. Динамическое включение fieldset

Проблема

Необходимо исключить поля из элемента **fieldset** и динамически включать их, если выполняется некоторое условие, например пользователь выбирает положение переключателя или устанавливает флажок.

Решение

Добавьте элементы **fieldset** и **legend** для группы связанных полей формы, которые будут скрыты. Обратите внимание, что элементы **fieldset** могут быть вложенными.

Добавьте атрибут **disabled** для двух вложенных элементов **fieldsets**. Для того чтобы применять изменения, укажите для каждого переключателя событие **onchange**:

```
<form>
    <fieldset>
        <legend>Кредитная карта</legend>
        <p><label>Имя, написанное на карте: <input name="fullName"
            required></label></p>
    <fieldset name="accountNum" disabled>
        <legend>
            <label>
            <input type="radio" name="accountType"
                    onchange="form.accountNum.disabled = !checked;
                    form.accountLetters.disabled=checked">Мой счет
                    из 12 цифр
            </label>
        </legend>
        <p><label>Номер карты: <input name="cardNum"
            required></label></p>
    </fieldset>
    <fieldset name="accountLetters" disabled>
        <legend>
            <label>
                <input type="radio" name="accountType"
                        onchange="form.accountLetters.disabled =
                        !checked; form.accountNum.disabled=checked">Мой
                        счет содержит буквы и числа
            </label>
        </legend>
        <p><label>Код карты: <input
            name="cardLetters"></label></p>
        </fieldset>
    </fieldset>
</form>
```

Если положение переключателя не выбрано, то все элементы, вложенные внутрь **fieldset**, будут отключены.

Выбор положения переключателя удаляет атрибут **disabled** в **fieldset**, чтобы сделать связанные поля доступными для редактирования, и добавляет атрибут **disabled** для элемента **fieldset**, связанного с другим переключателем.

Обсуждение

Есть и другие способы динамически включать поля формы, но этот метод — одно из будущих возможных преимуществ новых атрибутов **disabled**, добавленных для элементов **fieldset** в HTML5.

В зависимости от того, как пользовательские агенты поддерживают **disabled**, вам может понадобиться добавить сценарии, которые динамически устанавливают его для **fieldset**, вместо того чтобы жестко прописывать атрибут.

Дополнительная информация

Издание W3C о HTML5 для авторов веб-проектов с формами: http://dev.w3.org/html5/spec-author-view/forms.html.

7.8. Определение обязательных полей формы

Проблема

Необходимо визуально и программно обозначить для пользователей обязательные для заполнения поля формы.

Решение

Если все поля обязательны для заполнения, добавьте в верхней части формы инструкцию, чтобы сообщить об этом пользователям. Такой подход прекрасно работает для маленьких форм.

Добавление required

Если вы хотите применить интернациональный вариант (IL8N) или вариант для пользователей с плохим зрением, поместите метку на отдельную строку. Добавьте **(Required)** для каждого элемента **label** обязательного поля:

```
<p><label for="fName">Имя (Required)</label></p>
<p><input type="text" id="fname"></p>
```

Если дизайн требует, чтобы метки и поля были на одной линии, добавьте изображение-значок в элемент **label**. Установите значение **Required** для атрибута изображения **alt**:

```
<label for="fName">
    <img src="required-icon.jpg" alt="Required">
    First Name
</label>
<input type="text" id="fname">
```

Использование ARIA

Если дизайн требует, чтобы метки и поля были на одной линии, добавьте изображение-значок в элемент **label**. Установите значение **Required** для атрибута изображения **alt**, задайте значение **presentation** для атрибута изображения **role**, а также укажите атрибут **aria-required="true"** в поле ввода.

Обязательные изображения-значки могут быть выровнены по левому краю — так информация воспринимается легче:

```
<label for="fName">
    <img src="required-icon.jpg" role="presentation" alt="Required">
    Имя
</label>
<input type="text" id="fname" aria-required="true">
```

Новые вспомогательные технологии, поддерживающие ARIA, игнорируют изображения-значки, потому что элементы с ролью **presentation** намеренно пренебрегают API. Тем не менее они должны распознавать атрибут **aria-required** и объявлять, что поле является обязательным.

Такое решение работает и для старых вспомогательных технологий, где не поддерживается ARIA и игнорируются все его атрибуты, а вместо этого для значка применяется значение **Required** атрибута **alt**.

Обсуждение

Пока для атрибута **required** элемента **input** будет обеспечена поддержка браузерами и вспомогательными технологиями, потребуется некоторое время.

В HTML5 меняется и смысл элемента **strong**. Целесообразно использовать его со звездочкой, указывающей на обязательные поля, так как они должны содержать значения, прежде чем пользователь отправит информацию.

Атрибут required. Вы, вероятно, хотели бы видеть HTML5-атрибут **required** в элементе **input**, но браузеры и вспомогательные технологии в настоящее

время его не поддерживают. Для зрячих пользователей можно применить CSS, чтобы задать внешний вид полей, или добавить стилизованную звездочку, чтобы посетители могли визуально определить на экране обязательные поля:

```
<label for="fName">
    <strong>*</strong>
    Имя
</label>
<input type="text" id="fname" required>Обсуждение
```

Дополнительная информация

«ARIA и прогрессивое совершенствование» от Дрека Феатерстона (Derek Featherstone) http://www.alistapart.com/articles/aria-and-progressive-enhancement и «Будущее веб-доступности: расширения HTML5-элемента input» http://webaim.org/blog/future-web-accessibility-html5-input-extensions.

7.9. Использование ARIA для динамических обновлений

Проблема

Необходимо сообщать пользователям, когда части веб-страницы начинают динамически обновляться.

Решение

Области страницы, которые могут динамически обновляться, называют в ARIA *активным регионом*.

Во-первых, назначьте для HTML-элементов атрибуты **aria-live** там, где изменяется контент или могут произойти обновления, и решите, когда должно появиться сообщение об обновлении.

Затем выберите стандартную роль для активного региона. Назначьте ее в родительском HTML-элементе, содержащем информацию, которая может измениться. Если для этой роли подходит поведение по умолчанию, вам не нужно будет указывать атрибуты:

```
<div role="alert">
```

Существуют следующие значения стандартной роли для активного региона.

- **alert** — используйте его для единичного срочного уведомления. Область обрабатывается как активный регион и обновляется немедленно. Предупреждения не получают фокус и поэтому не могут быть закрыты пользователем.
- **alertdialog** — этот тип предупреждения может получить фокус. Когда появляется предупреждение, вы должны автоматически установить фокус на активном элементе в диалоговом окне, например на кнопке OK. Для этого включите в активный элемент атрибут **aria-describedby**.
- **log** — используйте его для историй или журналов сообщений об ошибках: новая информация добавляется в конец журнала, а старая информация может исчезнуть.
- **marquee** — применяйте значение для котировки акций и рекламных баннеров. Этот тип предупреждения похож на журнал, поскольку предоставляемая информация может часто меняться.
- **status** — используйте его для мелких уведомлений, не считающихся предупреждениями. Отмеченная область также реализуется как активный регион и не получает фокус. Если разные части страницы управляют изменением статуса, то для определения отношений используйте атрибут **aria-controls**.
- **timer** — применяйте значение для отображения времени, которое уже прошло или еще осталось. Таймер обновляется на фиксированный интервал, если стоит на паузе или достигает конца отсчета.

Если вам нужно нечто иное, нежели стандартная роль ARIA активного региона, вы можете создать собственную область для обновлений.

Пользовательские активные регионы. Во-первых, определите, где изменяется контент или могут произойти обновления, и решите, когда должно появиться сообщение об обновлении.

Далее установите атрибут **aria-live** для родительского элемента HTML, содержащего информацию, которая может измениться. Значение атрибута **aria-live** будет определять, насколько быстро обновление станет доступным для пользователей. Допустимы следующие значения:

- **aria-live="off"** — обновления не объявлены;
- **aria-live="polite"** — обновления будут объявлены, когда пользователь окажется в режиме ожидания или закончит свою текущую деятельность;
- **aria-live="assertive"** — обновления будут объявлены как можно скорее, даже если это чревато прерыванием текущей задачи пользователя.

Старайтесь не использовать **aria-live="assertive"**, если нет необходимости немедленно сообщить об изменениях. Пользователи могут посчитать такое вмешательство резким и грубым.

В вежливой беседе люди, чтобы вступить в разговор, ждут, пока наступит пауза. Так и **aria-live="polite"** сообщает об изменениях, когда пользовательская активность прерывается.

Итак, конкретнее рассмотрим элемент **div**:

```
<div aria-live="polite">
```

Следует решить, насколько здесь необходимо, чтобы пользователь смог принять обновления. Если имеет смысл применить изменения для всех активных регионов, назначьте для HTML-элемента атрибут **aria-atomic** со значением **true**.

Повторение информации без изменений может быть избыточно или усложнит поиск отличий. Если необходимо сообщить только об изменениях и они сами по себе имеют смысл, присвойте HTML-элементу атрибут **aria-atomic** со значением **false**:

```
<div aria-live="polite" aria-atomic="false">
```

Наконец, определите тип обновления, назначив соответствующий атрибут HTML-элемента. Типы обновлений:

- **relevant="additions"** — если в DOM добавляются новые узлы;
- **relevant="removals"** — когда узлы удаляются из DOM;
- **relevant="text"** — если в рамках существующих узлов изменяется текст.

Вы можете установить несколько значений для соответствующих атрибутов, добавив между значениями пробел:

```
<div aria-live="polite" aria-atomic="false" relevant="additions removals
    text">
```

По умолчанию используется вариант **relevant="additions text"**, что отражает наиболее распространенный тип изменений.

Обсуждение

Активные регионы ARIA представляют собой стандартный способ оповещения вспомогательных технологий о происходящих изменениях DOM и о том, как с ними справиться.

Вы можете предотвратить появление оповещений об обновлениях до тех пор, пока в активном регионе не будут применены все изменения. Для этого следует динамически настроить атрибут **state (aria-busy="true")**, а затем очистить его, когда можно будет объявить о готовности обновлений. Это может быть полезно при многочисленных обновлениях.

Дополнительная информация

Описание роли моделей WAI-ARIA 1.0 можно найти на странице http://www.w3.org/WAI/PF/aria/roles. Статья «Практика создания WAI-ARIA 1.0. Свойства и использование активных регионов» находится по адресу http://www.w3.org/WAI/PF/aria-practices/#liveprops. Несколько тестовых примеров активных регионов ARIA представлены на странице http://test.cita.ilinois.edu/aria/live/live1.php. Панель инструментов Juicy Studio Accessibility для Firefox для проверки активных регионов ARIA можно найти по адресу https://addons.moziila.org/en-US/firefox/addon/juicy-studio-accessibility-too/.

8 Геолокация

Кристофер Дойтч и Марк Грабански

8.0. Введение

API геолокации от W3C позволяет обращаться из кода сценариев к географической информации, предоставляемой браузерами пользовательских устройств. В этой главе мы сначала расскажем о том, как применять возможности подобного API, а затем рассмотрим на примерах, чего можно достичь, имея на руках геолокационные данные, Google Maps и геолокационные библиотеки других производителей, такие как SimpleGeo.

Во всех примерах главы применяется библиотека jQuery (http://jquery.com), которую можно легко подключить через элемент **script** двумя способами. Вообще, нет необходимости загружать файл **jquery.js** на локальный компьютер. Ваша веб-страница будет просто обращаться к версии, размещенной в сети Google Content Delivery Network (CDN):

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.js">
</script>
```

Однако на тот случай, если подключение к Интернету окажется недоступным, мы рекомендуем хранить локальную копию файла jQuery. Обращение к нему добавляется во второй элемент **script**, который сначала проверяет, была ли успешной попытка подключения в предыдущей строке:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.js">
</script>
<script>window.jQuery || document.write("<script
    src='js/libs/jquery-1.6.4.min.js'>\x3C/script>")</script>
```

8.1. Получение основных геолокационных данных

Проблема

Необходимо распознать местоположение пользовательского устройства, через которое посетитель вашего сайта вышел в Интернет.

Решение

Используйте новый API геолокации для получения данных о местоположении пользователя после того, как он нажмет соответствующую кнопку, а затем покажите географическую информацию на веб-странице (рис. 8.1).

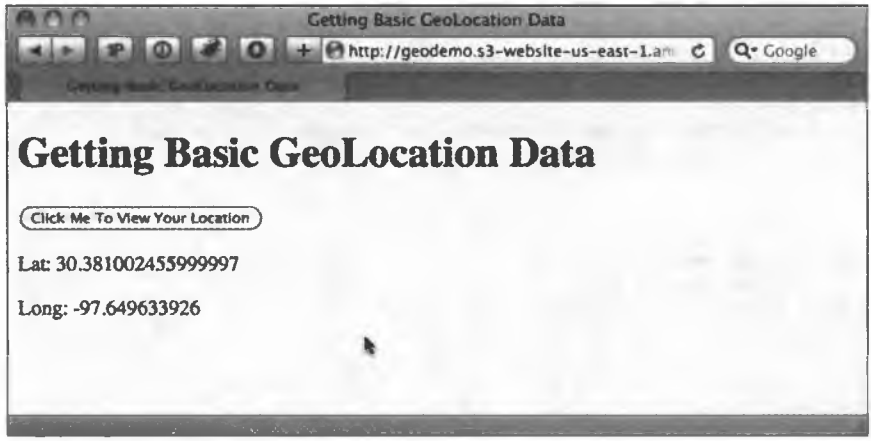

Рис. 8.1. Координаты широты и долготы определяются встроенными функциями браузера

Начните с добавления на страницу кнопки **input**:

```
<input type="button" id="go" value="Нажми меня, чтобы узнать свое местоположение" />
```

Теперь добавьте код JavaScript для обработки события **click** кнопки, обращения к API геолокации и вывода результатов:

```
<script>
$(document).ready(function () {
    // проверка нажатия кнопки
    $('#go').click(function () {
        // проверка наличия возможностей геолокации
```

продолжение ⇙

```
        if (navigator && navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(geo_success, geo_error);
        } else {
            error('Геолокация не поддерживается.');
        }
    });
});

function geo_success(position) {
    printLatLong(position.coords.latitude, position.coords.longitude);
}

// Возвращенный объект PositionError содержит следующие атрибуты:
// code: числовой код ответа
// PERMISSION_DENIED = 1
// POSITION_UNAVAILABLE = 2
// TIMEOUT = 3
// message: в основном предназначено для отладки. Не рекомендуется
// показывать данное сообщение пользователям.
function geo_error(err) {
    if (err.code == 1) {
        error('Пользователь запретил определение местоположения.')
    } else if (err.code == 2) {
        error('Географическая информация недоступна.')
    } else if (err.code == 3) {
        error('Превышен интервал ожидания географической информации.')
    } else {
        error('Во время запроса географической информации произошла
               неизвестная ошибка.')
    }
}

// вывод значений широты и долготы
function printLatLong(lat, long) {
    $('body').append('<p>Широта: ' + lat + '</p>');
    $('body').append('<p>Долгота: ' + long + '</p>');
}

function error(msg) {
    alert(msg);
}
</script>
```

Обсуждение

Объект **navigator** предоставляет доступ к новому объекту **geolocation**, у которого есть следующие методы:

- **getCurrentPosition()** — возвращает текущее местоположение пользователя;
- **watchPosition()** — возвращает местоположение пользователя, но продолжает наблюдать за его координатами и делает соответствующий обратный вызов каждый раз, когда местоположение меняется;

- **clearWatch()** — завершает мониторинг текущего местоположения, который осуществляется методом **watchPosition()**.

При определении местоположения устройства с выходом в Интернет в первую очередь проверяйте, что в браузере реализована встроенная поддержка возможностей геолокации. Если это так, то вызывайте метод **getCurrentPosition()**:

```
if (navigator && navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(geo_success, geo_error);
} else {
    rror('Геолокация не поддерживается.');
}
```

Поскольку этот метод выполняется асинхронно, нужно передавать ему две функции обратного вызова: **geo_success** и **geo_error**. Второму обратному вызову, предназначенному для ситуации с возникновением ошибки, передается объект **position error**, который содержит свойства **code** и **message**. Свойство **code** может принимать следующие значения:

- **0** — неизвестная ошибка;
- **1** — доступ запрещен;
- **2** — данные о местоположении недоступны;
- **3** — превышен тайм-аут.

При удачном выполнении метода обратному вызову **geo_success** передается объект **position**, включающий объект **coordinates** и метку времени. Объект **coordinates** предоставляет следующие данные:

- **latitude** — широта, выраженная в десятичных градусах;
- **longitude** — долгота, выраженная в десятичных градусах;
- **altitude** — высота в метрах над эллипсоидом;
- **accuracy** — точность в метрах;
- **altitudeAccuracy** — точность определения высоты в метрах;
- **heading** — направление движения, выраженное в градусах;
- **speed** — скорость в метрах в секунду.

Из перечисленных семи свойств гарантируется поддержка только трех: **latitude**, **longitude** и **accuracy**.

Для решения нашей задачи добавьте в тело веб-страницы значения **latitude** и **longitude**, воспользовавшись библиотекой jQuery:

```
function printLatLong(lat, long) {
    $('body').append('<p>Широта: ' + lat + '</p>');
    $('body').append('<p>Долгота: ' + long + '</p>');
}
```

Дополнительная информация

Спецификация API геолокации на сайте W3C: http://dev.w3.org/geo/api/spec-source.html.

8.2. Альтернативный способ получения основных геолокационных данных

Проблема

Необходимо определить местоположение пользователя, чей браузер не поддерживает API геолокации HTML5.

Решение

В качестве альтернативного варианта выполните поиск местоположения по IP-адресу. Разумеется, точность будет гораздо ниже, чем при определении координат широты и долготы, но это все же лучше, чем ничего.

Google или MaxMind

Google в своей библиотеке V3 API Google Maps предлагает объект **google.loader.ClientLocation**, который, к сожалению, не поддерживает многие IP-адреса в США.

Веб-служба GeoIP MaxMind, работающая на базе JavaScript, обеспечивает более высокую точность и предоставляет новейшие данные. Кроме того, ее можно использовать бесплатно, если добавить на страницу ссылку на сайт http://www.maxmind.com. Создатели MaxMind также предлагают не требующую указания авторства лицензию JavaScript, которую можно приобрести за $250 в год.

Кодирование решения

Изменим предыдущий пример, добавив альтернативный вариант определения местоположения на основе MaxMind. Начните с добавления на страницу библиотеки JavaScript:

```
<script src="http://j.maxmind.com/app/geoip.js"></script>
```

После этого добавьте альтернативное решение MaxMind:

```
$(document).ready(function () {
    // проверяем нажатие кнопки
    $('#go').click(function () {
```

```
        // проверяем наличие поддержки геолокации
        if (navigator && navigator.geolocation) {
            // делаем запрос местоположения пользователя
            navigator.geolocation.getCurrentPosition(geo_success, geo_error);
        } else {
            // используем альтернативный вариант: API MaxMind для
            // определения местоположения по IP-адресу
            printLatLong(geoip_latitude(), geoip_longitude(), true);
        }
    });
});

// выводим значения широты и долготы
function printLatLong(latitude, longitude, isMaxMind) {
    $('body').append('<p>Широта: ' + latitude + '</p>');
    $('body').append('<p>Долгота: ' + longitude + '</p>');
    // если мы определили местоположение с помощью MaxMind, необходимо
    // добавить ссылку на сайт разработчиков
    if (isMaxMind) {
        $('body').append('<p><a href="http://www.maxmind.com" target="_blank">IP
        to Location Service Provided by MaxMind</a></p>');
    }
}

function geo_error(err) {
    // вместо того чтобы отображать сообщение об ошибке, следует вызвать
    // библиотеку MaxMind для определения местоположения по IP-адресу
    printLatLong(geoip_latitude(), geoip_longitude(), true);
}
```

Вызывая printLatLong() с помощью MaxMind, передавайте дополнительный параметр true.

Обсуждение

Вместо того чтобы выводить сообщение об ошибке, когда объект **navigator** или **navigator.geolocation** остается неопределенным, применяйте функции **geoip_latitude()** и **geoip_longitude()**, которые извлекают информацию о широте и долготе местоположения пользователя и предоставляются JavaScript-библиотекой MaxMind.

Если вы взглянете на исходный код MaxMind-файла **geoip.js**, то увидите, что эта библиотека уже преобразовала ваш IP-адрес в географические данные. MaxMind динамически создает этот файл, считывая IP-адрес, с которого выполняется HTTP-запрос, переводя его в местоположение на серверной стороне, а затем выводя результаты обратно.

Помимо широты и долготы, доступна и другая географическая информация, перечисленная в табл. 8.1.

Таблица 8.1. Примеры географических данных из файла geoip.js

Метод	Описание	Пример данных
geoip_country_code()	Код страны	US
geoip_country_name()	Название страны	United States
geoip_city()	Город	Minneapolis
geoip_region_code()	Регион	MN
geoip_region_name()	Название региона	Minnesota
geoip_postal_code()	Почтовый индекс	55401
geoip_area_code()	Телефонный код зоны	612
geoip_metro_code()	Код округа	613

Бесплатная версия MaxMind требует указания авторства в форме ссылки на сайт библиотеки, поэтому в функцию **printLatLong()** добавлен параметр **isMaxMind**. Он указывает, что для определения местоположения была использована именно библиотека MaxMind:

```
function printLatLong(latitude, longitude, isMaxMind) {
    $('body').append('<p>Широта: ' + latitude + '</p>');
    $('body').append('<p>Долгота: ' + longitude + '</p>');
    // если для определения местоположения использовалась библиотека MaxMind,
    // следует добавить ссылку на сайт авторов
    if (isMaxMind) {
        $('body').append('<p><a href="http://www.maxmind.com" target="_blank">IP
            to Location Service Provided by MaxMind</a></p>');
    }
}
```

Необходимо также учесть ситуации, когда пользователь отвергает запрос на определение его местоположения или возникает какая-либо ошибка. Для обработки подобных непредвиденных обстоятельств настройте обработчик geo_error так, чтобы он обращался к альтернативному варианту определения местоположения по IP-адресу, как рассказывается в следующем рецепте.

Поскольку мы добавили альтернативный вариант, основанный на библиотеке MaxMind и не зависящий от «родных» возможностей геолокации, наше решение теперь может работать в большом количестве браузеров и устройств.

Дополнительная информация

MaxMind предлагает бесплатные решения геолокации с открытым кодом, поддерживающие поиск по городу, стране и IP-адресу: http://www.maxmind.com/app/ip-location.

8.3. Получение адреса путем обратного геокодирования широты и долготы

Проблема

У вас есть широта и долгота, и вы хотели бы представить их в виде адреса, удобного для восприятия человеком.

Решение

Используйте JavaScript-API от Google Maps для преобразования данных широты и долготы в адрес (рис. 8.2).

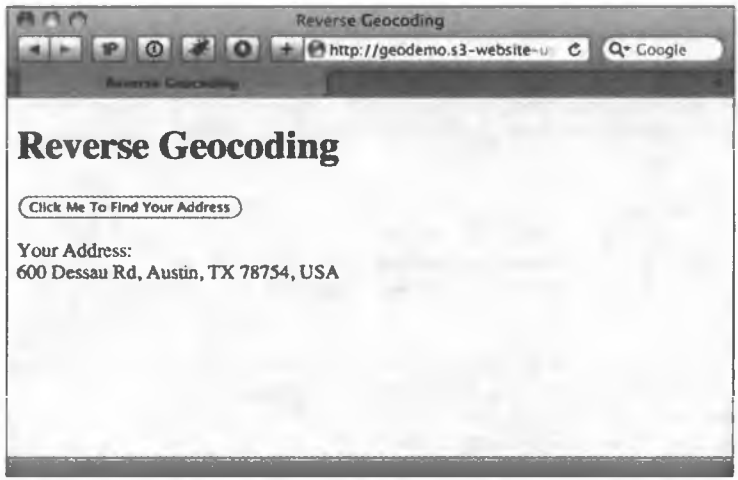

Рис. 8.2. Определение адреса пользователя на основе значений широты и долготы

Процесс преобразования таких географических данных, как почтовый адрес и индекс, в пространственные координаты, то есть значения широты и долготы, называется *геокодированием* (подробнее о нем мы поговорим в следующем рецепте). Обратная операция — превращение координат в адрес — носит название *обратного геокодирования*.

Для начала добавьте на свою веб-страницу необходимые сценарии:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.js">
</script>
<script src="http://j.maxmind.com/app/geoip.js"></script>
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

Определите кнопку, нажатие которой будет запускать процесс извлечения координат пользователя и поиск адреса:

```
<input type="button" id="go" value="Нажми меня, чтобы узнать свой адрес" />
```

Теперь добавьте JavaScript-код, обрабатывающий нажатие кнопки и обратный вызов **getCurrentPosition()** при успешном выполнении:

```
$(document).ready(function () {

    // проверяем нажатие кнопки
    $('#go').click(function () {
        // проверяем наличие поддержки геолокации
        if (navigator && navigator.geolocation) {
            // делаем запрос местоположения пользователя
            navigator.geolocation.getCurrentPosition(geo_success, geo_error);
        } else {
            // используем альтернативное решение: API MaxMind для
            // определения местоположения по IP-адресу
            printAddress(geoip_latitude(), geoip_longitude(), true);
        }
    });

function geo_success(position) {
    printAddress(position.coords.latitude, position.coords.longitude);
}

function geo_error(err) {
    // вместо того чтобы выводить сообщение об ошибке, обращаемся к библиотеке
    // MaxMind для определения местоположения по IP-адресу
    printAddress(geoip_latitude(), geoip_longitude(), true);
}

// используем API Google Maps для обратного геокодирования местоположения
function printAddress(latitude, longitude, isMaxMind) {
    // настраиваем объект Geocoder
    var geocoder = new google.maps.Geocoder();

    // преобразовываем координаты в объект
    var yourLocation = new google.maps.LatLng(latitude, longitude);

    // находим информацию о местоположении
    geocoder.geocode({ 'latLng': yourLocation }, function (results, status) {
        if (status == google.maps.GeocoderStatus.OK) {
          if (results[0]) {
            $('body').append('<p>Ваш адрес:<br />' +
                results[0].formatted_address + '</p>');
            } else {
            error('Google не возвратил результатов.');
            }
        } else {
            error("Обратное геокодирование не удалось, так как: " + status);
        }
    });
```

```
    // если мы применили MaxMind для определения местоположения,
    // нужно добавить ссылку на сайт авторов
    if (isMaxMind) {
        $('body').append('<p><a href="http://www.maxmind.com" target="_blank">IP
            to Location Service Provided by MaxMind</a></p>');
    }
}

function error(msg) {
    alert(msg);
}
```

Обсуждение

Получите координаты с помощью **getCurrentPosition()** и передайте их функции **printAddress()**, которая выполняет обратное геокодирование, обращаясь к API Google Maps.

Функция **printAddress()** начинает работу с создания нового объекта Google **Geocoder**. Он обеспечивает доступ к методу **geocode()**, который принимает различные параметры и, основываясь на переданных ему данных, возвращает информацию.

В нашем случае мы с помощью метода **google.maps.LatLng()** создаем новый объект Google **LatLng** и передаем его методу **geocode()** для получения почтового адреса. Метод **geocode()** выполняется асинхронно, как и **getCurrentPosition()**, поэтому мы определяем встраиваемую функцию JavaScript для обработки обратного вызова.

Итог обратного вызова состоит из двух параметров: в одном находится результат, а во втором — код **status**. Если значение **status** равно **OK**, то можно безопасно разбирать массив объектов **GeocoderResults** из переменной **results**. Она представляет собой массив, а не единичное значение, так как **Geocoder** поддерживает возвращение нескольких записей, а не только одной.

После этого мы проверяем наличие объекта **GeocoderResults** на первой позиции массива и, если он существует, присоединяем свойство **formatted_address** к телу веб-страницы.

Дополнительная информация

Подробнее об обратном геокодировании рассказывается на веб-странице http://code.google.com/apis/maps/documentation/javascript/services.html#ReverseGeocoding.

8.4. Преобразование адреса в широту и долготу

Проблема

Необходимо преобразовать почтовый адрес в широту и долготу.

Решение

Используйте JavaScript-API V3 от Google Maps для превращения данных адреса в широту и долготу, как показано на рис. 8.3. Этот процесс называется *геокодированием*.

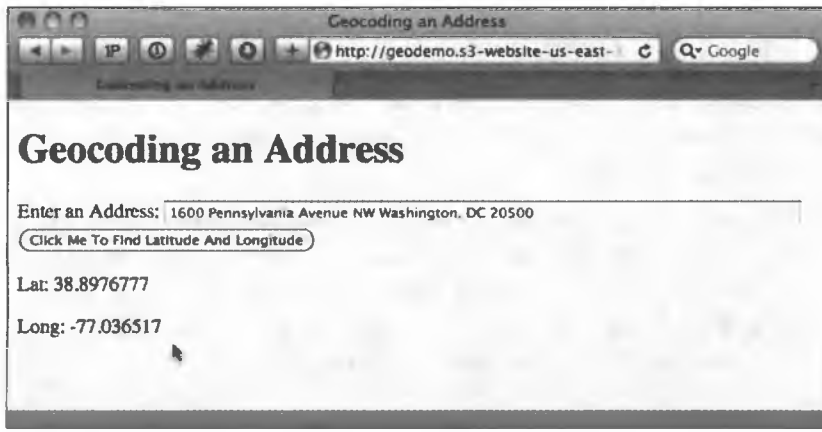

Рис. 8.3. Определение широты и долготы местоположения на основе адреса, указанного в понятной человеку форме

Для начала добавьте на веб-страницу библиотеку jQuery и JavaScript-API V3 от Google Maps:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.js">
</script>
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

API геолокации HTML5 умеет возвращать только координаты местоположения пользователя, поэтому придется самостоятельно ввести адрес в текстовом поле:

```
<div>
    <label for="address">Введите адрес:</label>
    <input type="text" id="address" />
</div>
```

```
<div>
    <input type="button" id="go" value="Нажми меня, чтобы узнать широту
и долготу"/>
</div>
```

В следующем коде JavaScript обрабатывается нажатие кнопки и считываются введенные пользователем данные, а также вызывается API Google для геокодирования адреса:

```
$(document).ready(function () {

    // проверяем нажатие кнопки
    $('#go').click(function () {
        // получаем введенный пользователем адрес
        var address = $('#address').val();
        if (address) {
            // используем API Google Maps для геокодирования адреса
            // настраиваем объект Geocoder
            var geocoder = new google.maps.Geocoder();
            // возвращаем координаты
            geocoder.geocode({ 'address': address }, function (results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
                if (results[0]) {
                // выводим результаты на экран
                printLatLong(results[0].geometry.location.lat(),
                    results[0].geometry.location.lng());
                } else {
                error('Google не возвратил результатов.');
                }
            } else {
                error("Геокодирование завершилось ошибкой, так   к: " + status);
            }
            });
        }
        else {
            error('Пожалуйста, введите адрес');
        }
    });
});

// выводим значения широты и долготы
function printLatLong(lat, long) {
    $('body').append('<p>Широта: ' + lat + '</p>');
    $('body').append('<p>Долгота: ' + long + '</p>');
}

function error(msg) {
    alert(msg);
}
```

Обсуждение

После того как пользователь нажмет кнопку, в работу включается библиотека jQuery, считывающая введенное значение и проверяющая, что оно не пустое. Затем создается экземпляр объекта **Geocoder**. Для этого нужно

вызвать метод **geocode()**, но на этот раз передать ему не значения широты и долготы, а адрес, указав для этого соответствующий параметр:

```
// настраиваем объект Geocoder
var geocoder = new google.maps.Geocoder();
// возвращаем координаты
geocoder.geocode({ 'address': address }, function (results, status) {
...
```

Теперь обратимся к свойству **geometry** объекта **GeocoderResults**. Оно, в свою очередь, содержит свойство **location**, с помощью которого можно вызывать методы **lat** и **lng** и получать соответствующие указанному адресу координаты. После того как координаты извлечены, их следует добавить к телу веб-страницы в функции **printLatLong()**:

```
// печатаем результаты
printLatLong(results[0].geometry.location.lat(), results[0].geometry.location.
lng());
```

Дополнительная информация

Подробнее о геокодировании рассказывается на веб-странице http://code.google.com/apis/maps/documentation/geocoding/.

8.5. Поиск маршрута на основе текущего местоположения

Проблема

Нужно определить маршрут, позволяющий добраться до определенной точки из текущего местоположения пользователя.

Решение

Используйте API Google Maps для отображения маршрута. Результат получается таким же, как на сайте Google Maps, и пользователь может самостоятельно выбрать единицы измерения расстояния: показывать его в милях или километрах (рис. 8.4).

Используйте jQuery и JavaScript-API V3 от Google Maps:

```
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script src="http://j.maxmind.com/app/geoip.js"></script>
```

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.js">
</script>
```

Рис. 8.4. Поиск дороги из одного местоположения в другое

Этот код HTML похож на тот, что мы приводили в предыдущем рецепте для реализации геокодирования. Добавляется только раскрывающийся список с возможностью выбора единиц измерения: мили или километры. Кроме того, вместо присоединения результатов к телу страницы мы добавляем один **div**-блок с вычисленным расстоянием и другой — для карты Google Maps:

```
<div class="field">
    <label for="address">Введите адрес назначения:</label>
```

продолжение ↛

```
    <input type="text" id="address" />
</div>

<div class="field">
    <label for="units">Единицы измерения:</label>
    <select id="units">
        <option value="IMPERIAL">мили</option>
        <option value="METRIC">километры</option>
    </select>
</div>

<div>
    <input type="button" id="go" value="Рассчитать маршрут" />
</div>

<div id="distance"></div>

<div id="map"></div>
```

Теперь потребуется код JavaScript, выполняющий следующие задачи:

- отображение карты США при загрузке страницы;
- обработка нажатия кнопки;
- получение текущего местоположения пользователя;
- считывание введенного адреса назначения;
- передача текущего местоположения и адреса назначения API Google для расчета длины автомобильного маршрута между двумя точками;
- обновление карты Google Maps и отображение предложенного маршрута.

Код выглядит так:

```
// глобальные переменные Google Maps:
var directionRenderer;
var directionsService = new google.maps.DirectionsService();
var map;

$(document).ready(function () {

    // Устанавливаем начальную точку для Google Maps.
    // Задаем начальные координаты: широта -92, долгота 32 – где-то около
    // Канзас-Сити в центре США. Устанавливаем уровень масштабирования,
    // равный 4, чтобы была видна вся территория США и чтобы материк
    // находился посередине области отображения.
    var kansas = new google.maps.LatLng(32, -92);
    var myOptions = {
        zoom: 4,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        center: kansas
    }
    map = new google.maps.Map(document.getElementById("map"), myOptions);
    directionsRenderer = new google.maps.DirectionsRenderer();
    directionsRenderer.setMap(map);
```

```
// проверяем нажатие кнопки
$('#go').click(function () {
    // Используем нашу новую функцию getLatLng с обратным вызовом
    // и определяем встраиваемую функцию для обработки обратного вызова.
    getLatLng(function (latitude, longitude, isMaxMind) {
        // устанавливаем начальную точку
        var start = new google.maps.LatLng(latitude, longitude);

        // получаем введенный пользователем адрес
        var address = $('#address').val();
        if (address) {
            // устанавливаем конечную точку
            var end = $('#address').val();

            // задаем параметры запроса
                var request = {
                origin: start,
                destination: end,
                travelMode: google.maps.DirectionsTravelMode.DRIVING
            };

            // запрашиваем маршрут
            directionsService.route(request, function (result, status) {
                if (status == google.maps.DirectionsStatus.OK) {

                    // Отображаем маршрут с помощью
                    // Directions Renderer от Google.
                    directionsRenderer.setDirections(result);

                    // отдельно выводим общую длину маршрута
                    var distance = getTotalDistance(result);
                    // выводим значение либо в милях, либо в километрах
                    var units = $('#units').val();
                    if (units == 'IMPERIAL') {
                        $('#distance').html('Общая длина маршрута: <strong>' +
                            metersToMiles(distance) + '</strong> миль');
                    } else {
                        $('#distance').html('Общая длина маршрута: <strong>' +
                            metersToKilometers(distance) + '</strong> км');
                    }
                } else {
                    error("Не удалось рассчитать маршрут, так как: " + status);
                }
            });

        }
        else {
            error('Пожалуйста, введите адрес');
        }

        // если для определения местоположения мы использовали MaxMind,
        // необходимо добавить ссылку на сайт авторов
        if (isMaxMind) {
            $('body').append('<p><a href="http://www.maxmind.com"
                target="_blank">IP to Location Service Provided by
                MaxMind</a></p>');
```

продолжение ⇩

```
            }
        });
    });
});

function getLatLng(callback) {
    // проверка наличия возможностей геолокации
    if (navigator && navigator.geolocation) {
        // запрашиваем местоположение пользователя
        navigator.geolocation.getCurrentPosition(function (position) {
            // обработчик для случая успеха
            callback(position.coords.latitude, position.coords.longitude);
        },
        function (err) {
            // обрабатываем ошибку, передавая обратному вызову данные
            // местоположения, полученные от MaxMind
            callback(geoip_latitude(), geoip_longitude(), true);
        });
    } else {
        // возможности геолокации недоступны, поэтому передаем обратному
        // вызову местоположение, полученное от MaxMind
        callback(geoip_latitude(), geoip_longitude(), true);
    }
}

// возвращаем общую длину маршрута в метрах
function getTotalDistance(result) {
    var meters = 0;
    var route = result.routes[0];
    for (ii = 0; ii < route.legs.length; ii++) {
        // Google хранит значение длины маршрута в метрах
        meters += route.legs[ii].distance.value;
    }
    return meters;
}

function metersToKilometers(meters) {
    return Math.round(meters / 1000);
}

function metersToMiles(meters) {
    // в 1 миле 1609,344 метра
    return Math.round(meters / 1609.344);
}

function error(msg) {
    alert(msg);
}
```

Обсуждение

Для реализации такого решения в первую очередь необходимо определить три глобальные переменные, с помощью которых будет происходить обмен данными с API Google и обновление **div**-блока **map**.

Мы проектируем решение так, чтобы при загрузке документа отображалась карта США. Карту на веб-страницах представляет объект Google **Map**, причем таких объектов может быть несколько.

Для создания объекта **Map** вызовите метод **new google.maps.Map(document.getElementById("map"), myOptions)**, передав ему HTML-элемент, в котором будет отображаться карта, и объект **Map options**.

Можно установить различные параметры вызова, однако мы использовали только три: **zoom**, **mapTypeId** и **center**. Названия параметров говорят сами за себя. Присвойте параметру **zoom** (масштаб) значение **4**, чтобы американский континент целиком поместился на карте. Параметру **mapTypeId** нужно присвоить значение **ROADMAP** (дорожная карта), устанавливающее отображение обычной двумерной карты Google Maps, которая состоит из фрагментов. Другие доступные значения — это **SATELLITE** (спутниковая), **HYBRID** (гибрид) и **TERRAIN** (топографическая). Параметр **center** позволяет указать, какое местоположение должно выводиться в центре карты.

Широта и долгота штата Канзас, находящегося в центре США, жестко закодированы при создании объекта **LatLng**, который мы и передаем параметру **center**. Когда объект **Map** создается с ключевым словом **new**, он автоматически обновляет наш **div**-блок **map**.

В следующей строке — **directionsRenderer = new google.maps.DirectionsRenderer();** — мы создаем новый объект **DirectionsRenderer**, который будет автоматически обновлять блоки **Map**. Строка **directionsRenderer.setMap(map);** предлагает пользователю ввести адрес и нажать кнопку.

В этом примере обратный геолокационный вызов описывается более компактно и может применяться в других задачах:

```
function getLatLng(callback) {
    // проверяем наличие возможностей геолокации
    if (navigator && navigator.geolocation) {
        // запрашиваем местоположение пользователя
        navigator.geolocation.getCurrentPosition(function (position) {
            // обработчик случая успеха
            callback(position.coords.latitude, position.coords.longitude);
        },
        function (err) {
            // обрабатываем ошибку, передавая обратному вызову местоположение,
            // определенное с помощью MaxMind
            callback(geoip_latitude(), geoip_longitude(), true);
        });
    } else {
        // возможности геолокации недоступны,  поэтому передаем обратному
        // вызову местоположение, определенное с помощью MaxMind
        callback(geoip_latitude(), geoip_longitude(), true);
    }
}
```

Функция **getLatLng()** принимает единственный параметр **callback**, определяющий обратный вызов и возвращающий переменные **latitude**, **longitude** и **isMaxMind**.

Мы проверяем существование объекта **navigator.geolocation** так же, как делали это раньше, но сейчас определяем для него встраиваемые обработчики обратного вызова, обращающиеся к общей функции **callback**. Она возвращает либо результат выполнения **getCurrentPosition()**, либо широту и долготу, полученные от MaxMind.

Что касается обработчика нажатия кнопки в главном примере, то в первую очередь мы вызываем новую функцию **get LatLng()** для получения текущего местоположения пользователя, которое затем применяется для формирования нового объекта **LatLng**, а его мы, в свою очередь, сохраняем в переменной **start**.

После этого мы считываем адрес и сохраняем его в форме строки в переменной **end**. Для получения маршрута применяется объект **DirectionsService**, заранее созданный и сохраненный в глобальной переменной **directionsService**. Метод **route()** объекта **DirectionsService** принимает в качестве параметров объект **DirectionsRequest** и метод обратного вызова. Объект **DirectionsRequest** поддерживает множество параметров, однако мы устанавливаем только три: **origin**, **destination** и **travelMode**.

Мы могли бы сделать запрос к API, чтобы выполнить геокодирование и получить координаты адреса, однако API Google автоматически делает это на следующем этапе.

Параметры **origin** и **destination** могут быть либо строками, как переменная **end**, либо значениями типа **LatLng**. Мы присваиваем параметру **travelMode** значение **DRIVING** (вождение автомобиля); два других доступных варианта — это **WALKING** (ходьба пешком) и **BICYCLING** (езда на велосипеде).

Метод **route()** выполняется асинхронно, поэтому мы определяем функцию обратного вызова и передаем ей объект **DirectionsResult** и код состояния **status**. Проверяем переменную **status**, чтобы убедиться в успешном завершении метода **route()**, а затем передаем объект **result** объекту **DirectionsRenderer**, обновляющему карту и выводящему подсвеченный маршрут между начальной и конечной точками путешествия.

Для того чтобы вы получили представление о содержимом переменной **result**, мы передаем ее функции **getTotalDistance()**, отвечающей за расчет длины маршрута. У объекта **result** есть свойство **routes**, представляющее собой массив объектов **DirectionsRoute**. Каждый объект — маршрут — описывает

один из вариантов, как из начальной точки попасть в конечную. По умолчанию возвращается один маршрут, если только вы не присвоили параметру **provideRouteAlternatives** значение **true**.

Наша функция **getTotalDistance()** проверяет лишь первый маршрут. У всех объектов **DirectionsRoute** предусмотрено множество свойств, однако нас интересует только **legs**. Это массив объектов **DirectionsLeg**, каждый из которых определяет один отрезок пути между начальной и конечной точками.

Если маршрут не содержит ни одной промежуточной точки, то есть ни одного значения **waypoint**, получается, он состоит из одного отрезка пути. Поскольку промежуточные точки мы не определяли, результат должен включать в себя единственный отрезок пути. Тем не менее мы на всякий случай проходим в цикле по всем потенциальным отрезкам.

Как и **route**, объект **leg** обладает множеством свойств, однако мы обращаемся только к свойству **distance**, описывающему объект **DirectionsDistance**. Свойство **value** объекта **DirectionsDistance** дает длину одного отрезка пути в метрах. В цикле мы складываем все отрезки и получаем общую длину маршрута в метрах.

Наконец, мы проверяем единицы измерения, выбранные пользователем в раскрывающемся списке, чтобы понять, в чем выводить расстояние: в милях или километрах. После этого мы вызываем одну из вспомогательных функций: **metersToKilometers()** или **metersToMiles()**. Они переводят метры в километры или мили соответственно и выводят результат в **div**-элементе **distance**.

Дополнительная информация

Подробнее о поиске маршрута с помощью API Google Maps рассказывается на сайте http://code.google.com/apis/maps/documentation/javascript/services.html#Directions.

8.6. Пример: определяем маршрут из Starbucks в Starbucks

Проблема

Вы хотели бы узнать дорогу от ближайшего кафе Starbucks к следующему ближайшему кафе Starbucks.

Решение

С помощью API Places от SimpleGeo найдите ближайшее к текущему местоположению пользователя кафе Starbucks. Когда координаты будут установлены, вызовите API SimpleGeo второй раз для поиска еще одного кафе Starbucks, находящегося ближе остальных к первому. Потом задействуйте API Google Maps, который поможет вам проложить маршрут между найденными точками.

Для начала добавьте API SimpleGeo к своему набору библиотек JavaScript:

```
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script src="http://j.maxmind.com/app/geoip.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.js">
</script>
<script src="http://cdn.simplegeo.com/js/1.2/simplegeo.places.jq.min.js">
</script>
```

API SimpleGeo предоставляется бесплатно, однако для получения ключа необходимо зарегистрироваться (http://simplegeo.com). После завершения регистрации выберите меню Tokens ▸ JSONP Tokens (Талоны ▸ JSONP-талоны), и там вы найдете нужный ключ API (рис. 8.5).

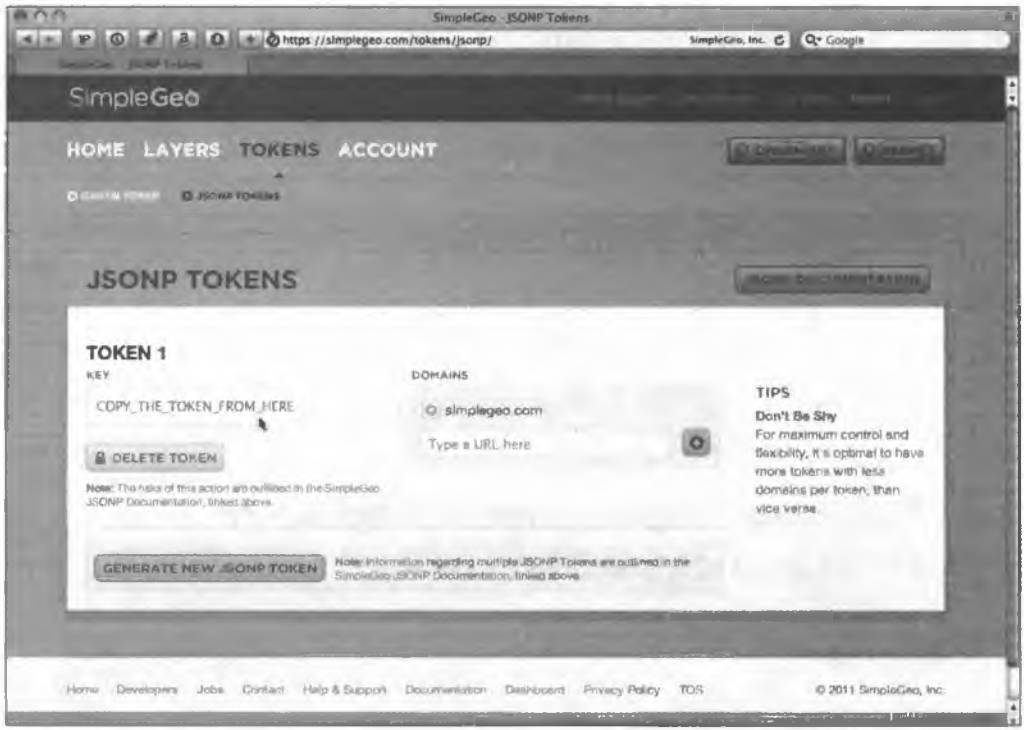

Рис. 8.5. Ключ API SimpleGeo

Добавьте домен своего сайта в список разрешенных доменов. Благодаря этому другие люди не смогут воспользоваться вашим ключом API. Теперь скопируйте ключ и вставьте его на место метки-заполнителя в начале кода JavaScript:

```
// глобальные переменные SimpleGeo:
var geoclient = new simplegeo.PlacesClient('УКАЖИТЕ СВОЙ КЛЮЧ API');

// глобальные переменные Google Maps:
var directionRenderer;
var directionsService = new google.maps.DirectionsService();
var map;
$(document).ready(function () {
  // Установка начальной точки для Google Maps.
  // Задаем начальные координаты: широта -92, долгота 32; где-то около
  // города Канзас-Сити в центре США. Затем устанавливаем уровень
  // масштабирования, равный 4, чтобы на карте континент
  // отображался целиком и находился в центре блока.
  var kansas = new google.maps.LatLng(32, -92);
  var myOptions = {
    zoom: 4,
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    center: kansas
  }
  map = new google.maps.Map(document.getElementById("map"), myOptions);
  directionsRenderer = new google.maps.DirectionsRenderer();
  directionsRenderer.setMap(map);

  // проверяем нажатие кнопки
  $('#go').click(function () {
    // Используем новую функцию getLatLng с альтернативным вариантом
    // поиска местоположения и определяем встраиваемую функцию
    // для обработки обратного вызова.
    getLatLng(function (latitude, longitude, isMaxMind) {
      // с помощью SimpleGeo находим ближайшее кафе Starbucks
      var query = "Starbucks";
      geoclient.search(latitude, longitude, { q: query, radius: 20,
          num: 1 }, function (err, dataStart) {
        if (err) {
          error(err);
        } else {
          // Мы запросили лишь один результат, а SimpleGeo возвращает
          // результаты, основываясь на расстоянии; поэтому ближайшая
          // точка идет первой. Проверим, что мы получили результат.
          if (dataStart.features.length == 1) {
            // сохраняем координаты начальной точки и адрес кафе
            var startLat =
              dataStart.features[0].geometry.coordinates[1];
            var startLng =
              dataStart.features[0].geometry.coordinates[0];
            var startAddress =
              dataStart.features[0].properties['address'];
            // также сохраняем в объекте Google LatLng
            var start = new google.maps.LatLng(startLat, startLng);
```

продолжение ↓

```
// Ищем кафе Starbucks, ближайшее к тому, которое
// мы только что нашли
geoclient.search(startLat, startLng, { q: query, radius:
    20, num: 2 }, function (err, dataEnd) {
  if (err) {
    error(err);
  } else {
    // На этот раз мы запросили два результата;
    // первый должен соответствовать тому кафе Starbucks,
    // которое мы приняли за начальную точку требуемого
    // маршрута, поэтому на этот раз мы
    // обращаемся ко второму результату.
    if (dataEnd.features.length == 2) {
      // сохраняем координаты конечной точки и адрес кафе
      var endLat =
        dataEnd.features[1].geometry.coordinates[1];
      var endLng =
        dataEnd.features[1].geometry.coordinates[0];
      var endAddress =
        dataEnd.features[1].properties['address'];
      // сохраняем также в объекте Google LatLng
      var end = new google.maps.LatLng(endLat, endLng);

      // Теперь добавляем маршрут, ведущий из первого кафе
      // Starbucks во второе.
      // Устанавливаем параметры запроса:
      var request = {
        origin: start,
        destination: end,
        travelMode:
        google.maps.DirectionsTravelMode.DRIVING
      };

      // запрашиваем маршрут
      directionsService.route(request, function
          (result, status) {
        if (status ==
          google.maps.DirectionsStatus.OK) {

          // Выводим маршрут с помощью
          // Directions Renderer от Google.
          directionsRenderer.
            setDirections(result);

          // по отдельности выводим данные
          $('#info').html('Ближайшее кафе Starbucks:
            <strong>' + startAddress
            + '</strong><br />' + 'следующее кафе
            Starbucks: <strong>' + endAddress
            + '</strong>');

        } else {
          error("Не удалось найти маршрут, так как: " +
            status);
        }
      });
    }
```

```
                else {
                  error('Не удалось найти кафе Starbucks рядом с ' +
                    startAddress);
                }
              }
            });
          }
          else {
            error('Не удалось найти кафе Starbucks рядом с вами.');
          }
        }
      });

      // если для определения местоположения мы использовали MaxMind,
      // необходимо добавить ссылку на сайт создателей
      if (isMaxMind) {
        $('body').append('<p><a href="http://www.maxmind.com"
          target="_blank">IP to Location Service Provided by
          MaxMind</a></p>');
      }
    });
  });
});

function getLatLng(callback) {
  // проверяем наличие возможностей геолокации
  if (navigator && navigator.geolocation) {
    // запрашиваем местоположение пользователя
    navigator.geolocation.getCurrentPosition(function (position) {
      // обработчик ситуации успеха
      callback(position.coords.latitude, position.coords.longitude);
    },
    function (err) {
      // обрабатываем ошибку, передавая обратному вызову местоположение,
      // определенное с помощью MaxMind
      callback(geoip_latitude(), geoip_longitude(), true);
    });
  } else {
    // геолокация недоступна, поэтому передаем обратному вызову
    // местоположение, определенное с помощью MaxMind
    callback(geoip_latitude(), geoip_longitude(), true);
  }
}

function error(msg) {
  alert(msg);
}
```

Обсуждение

Начало кода почти не отличается от предыдущего примера за исключением вызова **simplegeo.PlacesClient()** для настройки **SimpleGeo**.

В обработчике нажатия кнопки мы определяем текущее местоположение пользователя, применяя функцию **getLatLng()**, а затем полученные

значения широты и долготы задействуем при вызове **geoclient.search()**. На основании полученных координат функция **geoclient.search()** находит ближайшее кафе Starbucks. В качестве параметров она принимает значения широты и долготы, детали запроса и функцию обратного вызова.

Что касается деталей запроса, то в параметре **q** мы передаем критерий запроса, равный **Starbucks**; в параметре **radius** передаем радиус поиска, то есть 20 км, а также указываем в параметре **num**, что нам требуется только один результат.

Обратный вызов возвращает набор **features** с различной информацией о каждом результате поиска (включая широту и долготу). Здесь мы сохраняем широту, долготу и адрес в переменных, соответствующих начальной точке маршрута.

После этого мы второй раз вызываем **geoclient.search()**, передавая в качестве справочной точки начальную точку маршрута, то есть координаты первого найденного кафе. На этот раз функция возвращает два результата, причем в первом содержится местоположение изначального кафе. Значения широты, долготы и адреса из второго набора **feature** мы сохраняем в переменных, описывающих конечную точку маршрута.

Теперь, когда начальная и конечная точки маршрута заданы, мы можем прибегнуть к помощи тех же вызовов API Google Maps, которые применяли в предыдущем примере, и показать пользователю маршрут между двумя кафе на карте. В завершение мы выводим над картой адреса начальной и конечной точек маршрута.

Дополнительная информация

Создавайте динамические карты с наборами данных с помощью бесплатной библиотеки JavaScript, которую можно загрузить с сайта http://polymaps.org.

9 Элемент canvas

Кайл Симпсон

9.0. Введение

Одно из самых удивительных нововведений, стандартизированных в HTML5, — это элемент **canvas**, то есть *холст*:

```
<canvas id="mycanvas"></canvas>
```

Этот простой элемент создает на веб-странице прямоугольную блочную область, на которой вы (или ваши пользователи) можете рисовать такие объекты, как линии, окружности, и устанавливать заливки.

С элементом **canvas** связано множество полезных возможностей. Например, он поддерживает не только рисование вручную. Браузер умеет извлекать необработанные данные изображения из внешнего файла и «рисовать» их на элементе **canvas**.

Кроме того, можно легко очищать и перерисовывать как все изображение на элементе, так и его часть. Возможность редактирования естественным образом приводит к идее анимации, в которой впечатление движения создается за счет рисования, стирания и последующего перерисовывания объектов в новом положении.

Элементы **canvas** также поддерживают прозрачность, то есть объекты можно накладывать друг на друга для создания различных замысловатых изображений и графических эффектов. Предусмотрено несколько трансформаций и эффектов, которые можно применить к рисункам на элементе. По сути, изображение на элементе **canvas** — динамическое, а не статическое, как изображение из файла PNG или JPEG.

Аналогично тому, как изображение можно нарисовать в графическом редакторе, например Adobe Photoshop, сохранить в файл и загрузить на веб-страницу, рисование на элементе **canvas** можно запрограммировать. При этом результат будет появляться на веб-странице сразу после выполнения кода, и необходимость в загрузке внешнего файла отпадет.

Такая богатая функциональность позволяет реализовывать различные замысловатые эффекты, создавать которые раньше было слишком сложно или даже невозможно. Один из примеров — динамическое формирование эскизов для предварительного просмотра изображений.

Очень полезно понимать особенности работы напрямую с API canvas, о которых я и расскажу вам в этой главе. Однако на практике большинство специалистов при решении различных задач прибегают к помощи библиотек, автоматизирующих работу элемента canvas. Как вы увидите, некоторые программы оказываются довольно громоздкими, а использование библиотек помогает проектировщику не терять в производительности и сохранять рассудок.

9.1. Рисование на элементе canvas

Проблема

Необходим рисовать графические объекты прямо на странице, а не подгружать их как внешние изображения через элемент **img**.

Решение

Добавьте в разметку страницы элемент **canvas**:

```
<canvas id="mycanvas"></canvas>
```

Внутрь элемента canvas можно вставлять «запасное» содержимое, которое браузер будет показывать только в том случае, если canvas не поддерживается. Это не самый лучший механизм обеспечения доступности, однако это все же неплохая возможность добавить на веб-страницу альтернативный контент, который смогут распознать программы чтения экрана (подробнее о доступности контента для пользователей рассказывается в главе 7). Другой запасной вариант состоит в том, что canvas можно вставить внутрь элемента figure и добавить альтернативный текст с помощью элемента figcaption (см. рецепт 1.15). Подробнее о внедрении альтернативного контента для элемента canvas читайте на странице http://www.w3.org/TR/2dcontext/#focus-management.

Элемент **canvas** можно создать динамически, присоединить его к странице и с помощью CSS выбрать подходящее местоположение:

```
var mycanvas = document.createElement("canvas");
mycanvas.id = "mycanvas";
document.body.appendChild(mycanvas);
```

Чтобы нарисовать что-нибудь на элементе **canvas**, сначала извлеките ссылку на его контекст, а затем выполните команды рисования с использованием этой ссылки:

```
var mycanvas = document.getElementById("mycanvas");
var mycontext = mycanvas.getContext("2d");
mycontext.beginPath();
mycontext.moveTo(10, 10);
mycontext.lineTo(35, 35); // рисуем путь между точками (10,10) и (35,35)

mycontext.strokeStyle = "#000";
mycontext.stroke(); // рисуем линию

mycontext.beginPath();
mycontext.arc(35, 35, 10, 0, Math.PI * 2, true); // рисуем круг

mycontext.fillStyle = "#f00";
mycontext.fill(); // заливаем круг цветом
```

На рис. 9.1 показан результат выполнения этого кода.

Рис. 9.1. Элемент <canvas> на веб-странице; на элементе canvas нарисованы черная линия и красный круг

Очень важен порядок рисования объектов на элементе canvas, так как для каждого пиксела на странице отображается результат последней операции рисования. В этом примере мы сначала создали линию, а потом круг.

API canvas. Далее перечислены несколько часто используемых команд рисования из API элемента **canvas**.

- **beginPath()** — начало определения сегмента пути, который затем будет применяться в таких операциях визуализации, как **stroke()** и **fill()**.
- **closePath()** — закрытие любого пути: конец и начало пути соединяются прямым сегментом.
- **moveTo(*x*, *y*)** — перенос регистрационной точки для следующей относительной операции рисования.
- **lineTo(*x*, *y*)** — создание прямого пути между текущей регистрационной точкой и точкой **(*x*, *y*)**.
- **rect(*x*, *y*, *ширина*, *высота*)** — создание прямоугольного пути, один угол которого находится в точке **(x, y)**, а противоположный (по диагонали) — в точке **(*x*+*ширина*, *y*+*высота*)**.
- **arc(*x*, *y*, *радиус*, *начальный_угол_в_радианах*, *конечный_угол_в_радианах*, *против_часовой_стрелки*)** — создание пути в форме дуги (вплоть до полной окружности), где **(*x*, *y*)** — координаты центра дуги, которая начинается и заканчивается в точках, соответствующих указанным углам (в радианах). Начальная и конечная точка соединяются либо по часовой стрелке, либо против часовой стрелки.
- **fill()** — заполнение сегмента пути, определенного последним по времени.
- **stroke()** — обводка (то есть превращение в видимый объект) сегмента пути, определенного последним по времени.
- **drawImage(*изображение*, ...)** — рисование изображения в области, определяемой элементом **canvas**.
- **strokeText(*текст*, ...)** и **fillText(*текст*, ...)** — добавление на элемент **canvas** текста.
- **clearRect(*x*, *y*, *ширина*, *высота*)** — очистка прямоугольной области элемента **canvas**, определяемой углами **(*x*, *y*)** и **(*x*+*ширина*, *y*+*высота*)**.
- **strokeStyle=[*строка*|*объект*]** и **fillStyle=[*строка*|*объект*]** — установка атрибутов цвета и стиля для обводок и заливок соответственно.

Обсуждение

API элемента **canvas** для двумерного рисования стандартизован в HTML5 (http://dev.w3.org/html5/2dcontext/). По большей части, функциональность **canvas**

одинаково проявляется во всех современных браузерах, поддерживающих этот элемент. Однако существуют некоторые неявные различия между реализациями элемента в разных браузерах.

Поскольку отклонения в поведении элемента в разных браузерах и их версиях обычно не определяются в спецификациях и зависят исключительно от реализации, бесполезно описывать их в деталях или пытаться предсказывать.

Таким образом, мы не будем углубляться в эти особенности и лишь хотим попросить вас во избежание проблем тщательно тестировать работу элемента canvas и его API во всех возможных браузерах.

В спецификации элемента **canvas** описываются два *контекста визуализации*, то есть две системы координат. В настоящее время единственный реализованный контекст визуализации — это *2d*, определяющий плоскую двумерную декартову систему координат с началом отсчета (точкой с координатами (0, 0)) в верхнем левом углу (что соответствует системе координат веб-страниц). Практически все вызовы API **canvas** выполняются к этому контексту, а не к самому «холсту».

Предполагаем, что у вас уже есть ссылка на элемент **canvas**, поэтому для получения ссылки на его контекст 2d выполните следующую команду:

```
var mycontext = mycanvas.getContext("2d");
```

Имея в своем распоряжении контекст «холста», вы можете вызывать все команды, перечисленные в предыдущем разделе. Команды для рисования фигур в этом API основаны на путях. Иначе говоря, вы сначала «чертите» — или определяете, но не визуально — *путь* (один или несколько прямых или кривых сегментов), представляющий фигуру (линию, дугу/кривую, прямоугольник и т. д.), а затем указываете, что с ним нужно сделать.

Обычно путь после определения *обводят*, то есть рисуют вдоль него видимую линию. Можно также *залить* путь, то есть заполнить его внутреннюю область сплошным цветом или узором.

Поскольку путь визуализируется уже после того, как вы описали его контур, зачастую приходится создавать множество разных сегментов пути и применять к ним различные обводки и заливки, отображая каждый сегмент по мере определения.

Первый шаг в определении сегмента пути — это всегда вызов **beginPath()**:

```
mycontext.beginPath();
mycontext.moveTo(10, 10);
mycontext.lineTo(30, 30);
```

Если вызвать closePath() перед какой-либо командой визуализации, например stroke() или fill(), то путь в буквальном смысле будет «закрыт», то есть начальная точка пути автоматически соединится с конечной точкой последним прямым сегментом.

После определения сегмента пути вызовите команду визуализации **stroke()** или **fill()**. Они применяются к последнему определенному сегменту пути. Когда команды отработают, путь становится неактивным и недоступным:

```
mycontext.beginPath();
mycontext.moveTo(10, 10);
mycontext.lineTo(30, 30);
mycontext.stroke();
```

Учтите, что если вы нарисуете несколько фигур в составе одного сегмента пути, то для элемента **canvas** они будут частью единого целого. Имеется в виду, что в логике этого элемента конец одной фигуры соединяется с началом следующей прямым контуром. Однако это верно не для всех ситуаций: некоторые команды рисования фигур создают собственные независимые и не связанные с другими фигурами сегменты.

И снова повторимся: лучше явно определять все необходимые сегменты пути — это поможет избежать недоразумений. Один из таких явных способов включает использование команды **moveTo(...)**, которая виртуально поднимает перо и переносит его в новое место, прежде чем снова опустить на «холст». В противном случае возникает ситуация, аналогичная рисованию без отрыва пера: для перехода от одной точки к другой всегда приходится вычерчивать линию на «холсте», даже если речь идет о переходе от конца одной фигуры к началу другой.

Векторные и растровые изображения. Наверняка вы уже слышали о *векторной* и *растровой* (или битовой) графике. В векторной графике изображение описывается как последовательность фигур с помощью набора уравнений. Благодаря точному математическому определению эти фигуры (векторы) можно преобразовывать (поворачивать, масштабировать, трансформировать и т. д.) без потери качества.

В противоположность этому, растровая графика подразумевает закрашивание отдельных пикселов различными цветами. Преобразование растрового изображения ведет к потере качества: вдоль границ между цветными областями появляются расплывчатые или смазанные фрагменты, так как цвета смешиваются, то есть пикселы окрашиваются новым цветом, представляющим среднее значение цветов ранее соседних пикселов.

Работая с элементом **canvas**, важно не забывать о различиях между векторными и растровыми изображениями. Пути, которые вы определяете такими командами API, как **lineTo(...)**, подобны векторам: их можно рисовать

и модифицировать безо всякой потери качества. Указав путь подходящей формы, вы визуализируете его, то есть создаете видимые пикселы на растровом «холсте». Если после этого применить к растровой картинке дополнительные преобразования, например повернуть целый элемент, то это, скорее всего, приведет к потере качества, как описано выше.

Проще говоря, думайте об описаниях путей как о векторных уравнениях, которые нельзя увидеть, но можно менять по своему усмотрению, пока визуализация не выполнена. Однако операции обводки и заливки, применяемые к путям, — это уже визуализация растровых изображений. В целом, любой графический редактор, предназначенный для обработки векторной графики, работает именно так.

Если вы нарисуете фигуру (или измените местоположение или размеры объекта), пересекающую только часть пиксела, например линию с координатами (10.5, 20), (10.5, 50), то каждая половина пиксела будет визуализирована лишь частично (это называется сглаживанием), что приведет к появлению нечетких, расплывчатых областей.

Существуют операции, которые *всегда* применяются только к конечному визуализированному результату (например, изменение цвета). Они относятся к сфере растровой, или пиксельной графики, в противоположность геометрическим трансформациям, доступным для векторных путей.

Дополнительная информация

Подробнее об использовании элемента **canvas** рассказывается в учебнике на сайте MDC: https://developer.mozilla.org/en/Canvas_tutorial.

9.2. Использование эффекта прозрачности

Проблема

Необходимо нарисовать на элементе **canvas** фигуры с прозрачными областями, чтобы сквозь них можно было увидеть фон.

Решение

Элемент **canvas** по умолчанию прозрачен, то есть сквозь него просвечивает любой контент, находящийся ниже. Какие пикселы будут прозрачными, а какие — нет, вы определяете, просто рисуя на «холсте».

Если поместить рисунок с красным кругом (мы научились создавать его в предыдущем рецепте) поверх текста, то области элемента **canvas** за пределами этого круга останутся прозрачными и сквозь них можно будет увидеть текст (рис. 9.2).

Рис. 9.2. Рисование круга поверх страницы с текстом: слова просвечивают сквозь прозрачную область «холста»

Помимо этого, любые пикселы, нарисованные на элементе **canvas**, можно сделать частично прозрачными, отредактировав альфа-каналы цветов, которыми они закрашены. *Частичная прозрачность* означает, что пользователь увидит новый цвет, представляющий собой комбинацию цветов пикселов *под* элементом **canvas** и *на* нем.

Например, если сделать красный круг из предыдущего примера частично прозрачным, то результат будет выглядеть, как на рис. 9.3.

Частичная прозрачность полезна для оформления контента, не только находящегося на странице под элементом **canvas**, но и уже нарисованного на «холсте»:

```
mycontext.beginPath();
mycontext.arc(40, 40, 25, 0, Math.PI * 2, true); // рисуем круг
mycontext.closePath();

mycontext.fillStyle = "#f00";
mycontext.fill(); // заполняем круг сплошным цветом

mycontext.beginPath();
mycontext.arc(70, 40, 25, 0, Math.PI * 2, true); // рисуем другой круг
mycontext.closePath();

mycontext.fillStyle = "rgba(0,0,255,0.75)";
mycontext.fill(); // заливаем второй круг сплошным цветом
```

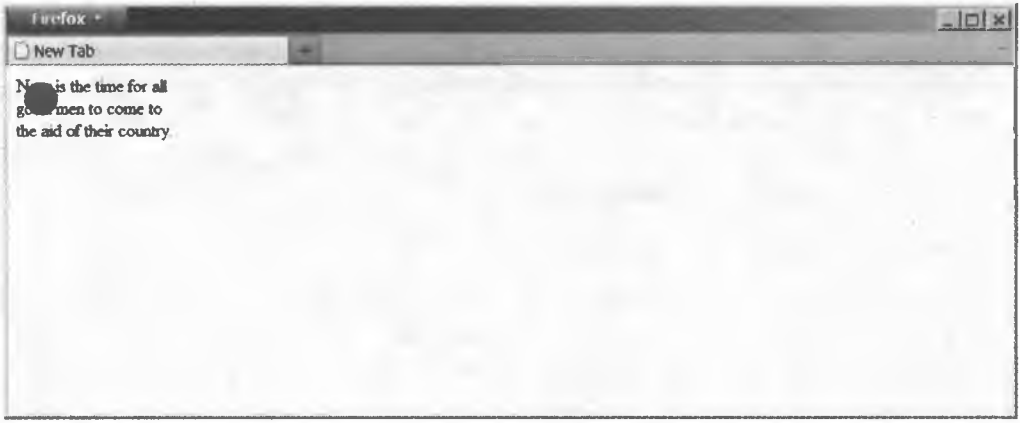

Рис. 9.3. Красный круг частично прозрачен и позволяет видеть расположенный под ним текст

В предыдущем фрагменте кода мы нарисовали частично прозрачный синий круг, пересекающийся с красным кругом. В области пересечения, где синий цвет накладывается на красный, появляется фиолетовый цвет (рис. 9.4).

Рис. 9.4. Мы сделали синий круг частично прозрачным, чтобы его цвет смешивался с цветом находящегося ниже красного круга

Обсуждение

Элемент **canvas**, на котором еще ничего не нарисовано, представляет собой область, содержащую только полностью прозрачные пикселы. Если на «холсте» нарисовать пиксел, залитый абсолютно непрозрачным цветом, то сквозь

него не будет просвечивать контент, расположенный ниже. В то же время частичная непрозрачность цвета означает, что контент будет просвечивать в некоторой степени. По умолчанию используются абсолютно непрозрачные цвета. Прозрачность в альфа-канале цвета определяется в формате RBGA, как и цветовые значения в таблицах стилей CSS3 (http://www.w3.org/TR/2003/CR-css3-color-20030514/#numerical), например:

```
mycontext.fillStyle = "rgba(255,0,0,0.5)";
```

Здесь значение **0.5** указывает на 50%-ю непрозрачность красного цвета, то есть все пикселы, закрашенные этим цветом, будут наполовину прозрачными, и контент, лежащий под ними, будет частично просвечивать (рис. 9.5).

Еще один способ настроить прозрачность визуализируемых пикселов — установить свойство **globalAlpha**, присвоив ему значение от **0.0** до **1.0**.

Рис. 9.5. Задание 50%-й прозрачности цвета для заливки красного круга позволяет увидеть черную линию, нарисованную под ним

Когда пиксел на «холсте» закрашивается цветом с частичной прозрачностью, то по умолчанию выполняется следующая операция: существующий цвет пиксела (если он задавался ранее) смешивается с новым цветом, что в результате дает третий цвет. Именно благодаря этому нам кажется, что на рис. 9.5 черная линия находится под частично прозрачным красным кругом.

Необходимо помнить, что в действительности мы имеем дело не с разными слоями, на которых находятся отдельные фигуры (как это бывает в графических редакторах), хотя впечатление создается именно такое. При визуализации пикселов цветовая информация объединяется: значение, которое было

определено для первого пути, объединяется со значением, определенным для второго пути. В конечном итоге мы все равно работаем с однослойным пиксельным изображением внутри элемента **canvas**. Несмотря на то что две фигуры визуализировались последовательно, одна над другой, невозможно переместить их или изменить независимо друг от друга, так как они являются частью одного растрового изображения.

Тем не менее поддержка прозрачности для пикселов элемента **canvas** позволяет имитировать слои: нужно всего лишь наложить друг на друга несколько элементов **canvas** и нарисовать на каждом из них свои фигуры. В таком случае не будет объединения независимых слоев (так как они определяются как отдельные элементы, хотя на уровне браузера, операционной системы и экрана компьютера наложение все равно произойдет), и разные элементы можно будет перемещать, изменять, стирать и поворачивать независимо друг от друга. Об этом рассказывается в рецепте 9.10.

Дополнительная информация

Подробнее о прозрачности вы прочтете в учебнике по стилям и цветам «холста» на сайте MDC: https://developer.mozilla.org/En/Canvas_tutorial/Applying_styles_and_colors.

9.3. Установка размеров «холста»

Проблема

Необходимо явно указать ширину и высоту элемента **canvas**, отличающиеся от размеров по умолчанию.

Решение

Добавьте к элементу **canvas** атрибуты **width** и **height** и присвойте им нужные значения:

```
<canvas id="mycanvas" width="200" height="200"></canvas>
```

Кстати, ширину и высоту элемента **canvas** можно менять и в коде JavaScript. Для того чтобы определить размер области визуализации «холста» (то есть, фактически, количество пикселов на ней), необходимо задать значения

атрибутов элемента **canvas**, а не свойств **width** и **height** стиля CSS, как может показаться:

```
mycanvas.setAttribute("width", "200"); // меняем размер в пикселах
mycanvas.setAttribute("height", "200");
```

Свойства **width** и **height** можно определять и напрямую в элементе:

```
mycanvas.width = 200; // меняем размер в пикселах
mycanvas.height = 200;
```

В любом из примеров выше элемент **canvas** сможет использовать для визуализации 200 пикселов по горизонтали и 200 пикселов по вертикали.

В противоположность этому, управление размером элемента **canvas** с помощью CSS — либо через правила, либо непосредственно путем установки свойств в JavaScript — не влияет на количество пикселов «холста». Существующий элемент **canvas** (с его пиксельными размерами) просто растягивается или сжимается, то есть меняются только его физические параметры:

```
mycanvas.style.width = "200px"; // уменьшаем визуальный размер по горизонтали
mycanvas.style.height = "200px";// увеличиваем визуальный размер по вертикали
```

Обсуждение

Размер элемента **canvas** по умолчанию — 300 пикселов по ширине и 150 пикселов по высоте. На практике разработчики чаще всего устанавливают другие размеры «холста».

Как и при работе с любыми другими блочными элементами HTML, если вы задаете абсолютную позицию элемента canvas, это еще не означает, что у него автоматически появляются реальные физические размеры, которые можно оценить по картинке на экране. В коде CSS необходимо явно задавать физические габариты «холста» в дополнение к его пиксельным размерам. Для того чтобы картинка не искажалась, всегда проверяйте, что физические размеры «холста» соответствуют пиксельным.

Пиксельные размеры «холста» определяются атрибутами элемента **canvas** — либо в разметке страницы, либо через вызов **setAttribute(...)**. Размер «холста» можно поменять, настроив стили CSS для его ширины и высоты, однако при этом растягивается или сжимается физическое представление элемента **canvas**, а количество пикселов в нем остается постоянным.

Например, вам требуется полностраничный элемент **canvas**, размер которого будет меняться вместе с размером окна браузера. Для того чтобы добиться

этого эффекта с сохранением пиксельных размеров «холста», используйте следующий CSS-код:

```
#mycanvas { width:100%; height:100%; }
```

Однако для определения элемента **canvas**, у которого при увеличении или уменьшении окна браузера будут меняться пиксельные размеры, вам потребуется код JavaScript:

```
window.onresize = function() {
    mycanvas.width = document.documentElement.clientWidth;
    mycanvas.height = document.documentElement.clientHeight;
};
```

Размеры элемента **canvas** можно менять так часто, как это требуется. Однако при каждом изменении область рисования очищается. При необходимости можно быстро очистить «холст», просто присвоив его атрибуту **width** текущее значение:

```
function clear(mycanvas) {
    mycanvas.width = mycanvas.width;
}
```

Дополнительная информация

Подробнее об использовании «холста» рассказывается в учебнике по элементу **canvas** на сайте MDC: https://developer.mozilla.org/en/Canvas_tutorial.

9.4. Использование градиентов, узоров и стилей линий

Проблема

Вы хотите создавать на своих страницах градиенты и использовать разные стили рисования.

Решение

Каждый раз, когда путь визуализируется для отображения на «холсте», для рисунка применяются установленные на данный момент цвет и стиль обводки и заливки.

Например, оформление сегментов линии можно настраивать благодаря таким стилям обводки, как **lineWidth**, **lineCap** и **lineJoin**:

```
mycontext.lineWidth = "12";
mycontext.lineJoin = "round";
mycontext.moveTo(20, 20);
mycontext.lineTo(50, 50);
mycontext.lineTo(20, 70);
mycontext.stroke();
```

Результат выполнения этого кода показан на рис. 9.6.

Рис. 9.6. Использование стилей обводки lineWidth и lineJoin

Стили заливки позволяют создавать градиенты или даже узоры внутри путей, которые вы рисуете на «холсте» (рис. 9.7):

```
var lingrad = mycontext.createLinearGradient(20,20,40,60);
lingrad.addColorStop(0.3, "#0f0");
lingrad.addColorStop(1, "#fff");
mycontext.fillStyle = lingrad;

mycontext.moveTo(20, 20);
mycontext.lineTo(50, 50);
mycontext.lineTo(20, 70);
mycontext.closePath();
mycontext.fill();
```

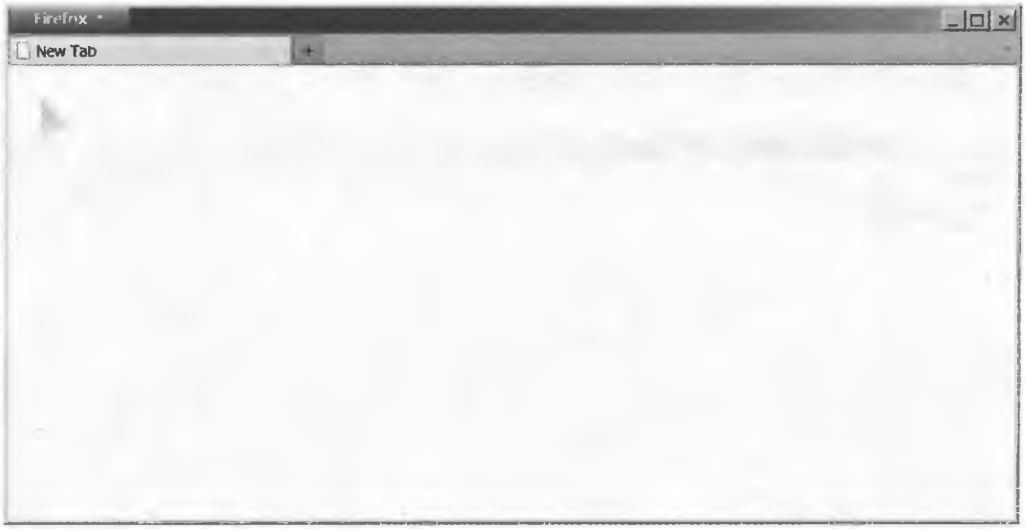

Рис. 9.7. Фигура, залитая линейным градиентом

Обсуждение

Стилями визуализации путей можно управлять разными способами. Например, ширина линий (обводок) устанавливается в свойстве **lineWidth**. Свойство **lineCap** управляет визуализацией концов линии (срезанный, скругленный и т. д.), а свойство **lineJoin** — формой соединения двух сегментов линии. На рис. 9.8 показано несколько разных стилей линий.

Помимо стилей, устанавливающих форму линий, для обводок и заливок поддерживаются различные цветовые оформления. Есть два основных варианта: градиенты и узоры.

Градиент — это плавный переход одного цвета в другой (может включать и больше двух цветов). Градиенты бывают *линейные* (цвет изменяется вдоль прямой линии) и *радиальные* (цвет меняется кругообразно, от центра окружности к ее краю). Для добавления градиента на веб-страницу нужно создать сам объект градиента, определить для него одну или несколько *цветовых остановок*, а после этого применить объект в качестве **strokeStyle** или **fillStyle**.

Узоры — это, по сути, любые изображения (или даже другие элементы **canvas**), которые применяются в качестве «цветов» линий или заливок, причем изображение может повторяться и укладываться внутри выбранной области различными способами. Вы создаете объект узора, указываете, какое изображение использовать и каким способом его повторять, а затем

устанавливаете его в качестве **strokeStyle** или **fillStyle**. На рис. 9.9 показан результат применения нескольких разных градиентов и узоров заливки.

Рис. 9.8. Различные стили линий, нарисованных на «холсте»

Рис. 9.9. Заливка фигур различными градиентами и узорами

При настройке заливки узором можно определить способ повторения (выкладки) изображения. Команда API «холста» **createPattern(...)** принимает второй, строковый параметр, который и управляет повторением изображения. Если повторять изображение не требуется, используйте значение **no-repeat**. В настоящее время единственный поддерживаемый во всех браузерах вариант выкладки — это **repeat**. Он определяет равномерное повторение изображения по горизонтали и по вертикали (рис. 9.10):

```
var img = document.getElementById("my_fish_image");
var imgfill = mycontext.createPattern(img, "repeat");
mycontext.fillStyle = imgfill;
mycontext.fillRect(0, 0, 200, 200);
```

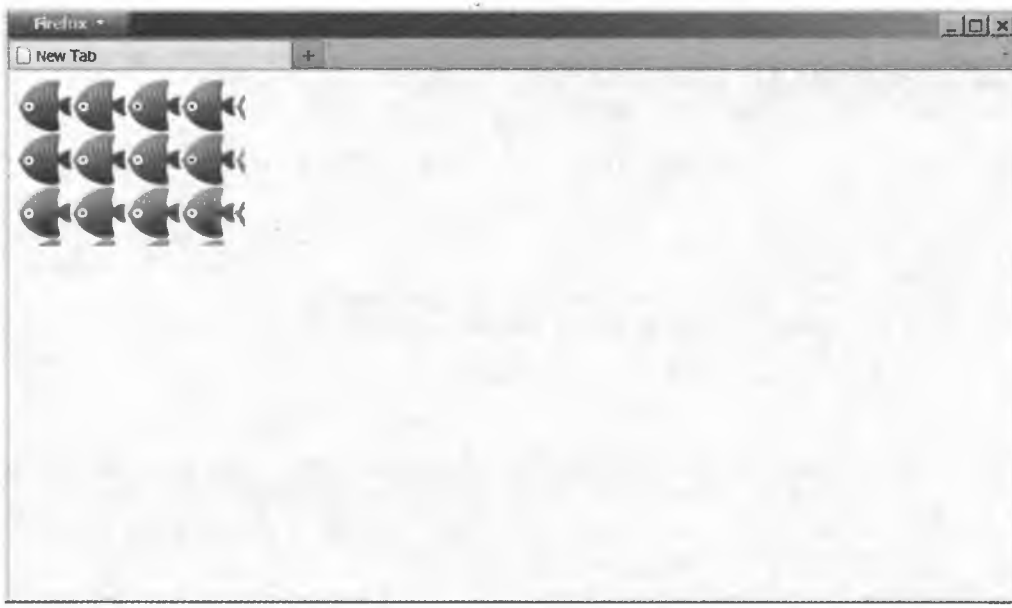

Рис. 9.10. Создание изображений с помощью узора заливки

Дополнительная информация

Подробнее о стилях линий, градиентах и заливках узорами рассказывается в учебнике по стилям и цветам «холста» на сайте MDC: https://developer.mozilla.org/En/Canvas_tutorial/Applying_styles_and_colors.

9.5. Добавление на «холст» внешних изображений

Проблема

У вас есть изображение (диаграмма, значок, фотография и т. п.) во внешнем файле, и вы хотите поместить его на элемент **canvas**, чтобы как-то изменить или украсить прямо на «холсте» с помощью команд рисования.

Решение

Сначала загрузите изображение в коде веб-страницы, используя элемент **img**, и извлеките ссылку на него. Можете также динамически создать элемент **Image** с помощью кода JavaScript:

```
var img = new Image();
img.src = "http://somewhere/to/my/image.jpg";
```

Затем, когда изображение будет загружено, поместите его на элемент **canvas**:

```
var img = new Image();
img.onload = function() {
    // примечание: здесь мы используем вызов к контексту 2d
    mycontext.drawImage(img, 0, 0); // рисуем изображение в точке (0,0)
};
img.src = "http://somewhere/to/my/image.jpg";
```

Объект, который вы передаете в качестве первого параметра функции drawImage(...), может быть статическим изображением, еще одним элементом canvas или элементом video (когда кадр видеоролика захватывается и помещается на «холст»).

Предполагается, что правильный размер элемента **canvas** установлен заранее и нужное изображение умещается на «холсте». Если «холст» оказывается слишком маленьким по высоте или ширине, то изображение обрезается снизу или справа соответственно.

Обсуждение

API «холста» предоставляет команду **drawImage(...)**, позволяющую захватить данные растрового изображения из иного источника (файла изображения, другого элемента **canvas**, элемента **video**) и поместить их прямо на элемент **canvas**.

Функцию **drawImage(...)** можно вызвать тремя способами. Самый простой вариант, показанный выше, принимает в качестве параметров элемент, из которого извлекаются данные растрового изображения, и две координаты, то есть точку (*X*, *Y*) внутри «холста», куда будет помещен верхний левый угол изображения.

Второй вариант принимает также два дополнительных параметра — (*dw*, *dh*), представляющих ширину и высоту области исходного изображения, которую вы желаете захватить и скопировать на «холст».

Третья форма вызова принимает еще больше параметров. Первые четыре числовых параметра — (*sx*, *sy*, *sw*, *sh*) — указывают местоположение верхнего левого угла и ширину/высоту области исходного изображения, которая будет захвачена и скопирована на «холст». Вторая четверка параметров — (*dx*, *dy*, *dw*, *dh*) — представляет координату верхнего левого угла и ширину/высоту области «холста», в которую функция **drawImage(...)** поместит захваченное изображение:

```
function createCanvas(id, width, height) {
    var canvas = document.createElement("canvas");
    canvas.id = id;
    canvas.setAttribute("width", width);
    canvas.setAttribute("height", height);
    document.body.appendChild(canvas);
    return canvas;
}

var small_canvas = createCanvas("small", 100, 100);
var large_canvas = createCanvas("large", 300, 300);
var small_context = small_canvas.getContext("2d");
var large_context = large_canvas.getContext("2d");

var img = new Image();
img.onload = function() {
    // примечание: здесь мы используем разные значения (sw,sh) и (dw,dh),
    // чтобы сжать изображение
    small_context.drawImage(img, 0, 0, 300, 300, 0, 0, 100, 100);
    // захватываем только область размером 300×300 пикселов,
    // начиная с верхнего левого угла изображения
    large_context.drawImage(img, 0, 0, 300, 300);
};
img.src = "http://somewhere/to/my/image.jpg";
```

Понять назначение описанных параметров можно из рис. 9.11.

Данные изображения, скопированного на «холст» таким способом, ничем не отличаются от данных любых других растровых изображений, которые вы могли бы нарисовать. Это означает, что с помощью описанных в предыдущих рецептах функций можно с легкостью нарисовать что-нибудь поверх этого изображения.

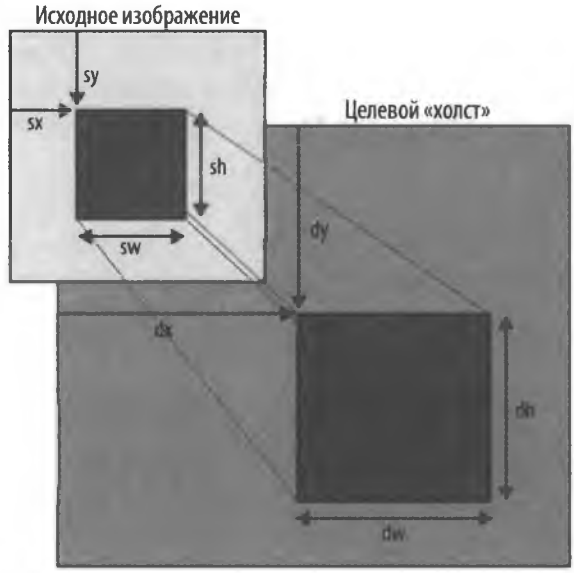

Рис. 9.11. Графическое пояснение параметров (dx, dy, dw, dh) и (sx, sy, sw, sh) функции drawImage(...)

Дополнительная информация

Руководство по копированию изображения на «холст» с помощью Ajax вы найдете на сайте http://www.html5canvastutorials.com/advanced/html5-canvas-load-image-data-url/.

9.6. Настройка цветовых преобразований

Проблема

Вы хотели бы изменить цветовую схему существующего рисунка на элементе **canvas**, например перевести его в оттенки серого или инвертировать цвета.

Решение

API **canvas** предоставляет команду **getImageData(...)**, которая захватывает цветовые данные всех пикселов «холста» и передает их в одном длинном массиве:

```
var drawing = mycontext.getImageData(0, 0, 200, 200);
```

Формат массива, который вы получаете от getImageData(...), может показаться несколько неожиданным. Это не двумерный массив, соответствующий ширине и высоте области «холста», а линейный одномерный массив, состоящий из следующих друг за другом «строк» изображения.

Помимо этого, для каждого пиксела изображения в массиве предусмотрены четыре последовательные записи, каждая из которых представляет один цветовой компонент. Таким образом, каждый пиксел занимает четыре «места» в массиве, формат которого таков: [..., красный, зеленый, синий, альфа, ...].

После того как данные изображения захвачены, их можно изменить по своему усмотрению и записать результат обратно в элемент **canvas**.

Чтобы перевести изображение в оттенки серого, проще всего присвоить каждому из трех цветовых компонентов значение в виде средней величины по красному, зеленому и синему цвету:

```
var avg;
// обрабатываем по 4 записи (1 px) за один раз
for (var i = 0; i < drawing.data.length; i = i + 4) {
    avg = (drawing.data[i] + drawing.data[i+1] + drawing.data[i+2]) / 3;
    drawing.data[i] = drawing.data[i+1] = drawing.data[i+2] = avg;
}
```

Если стоит задача инвертировать цвета изображения, то каждому цветовому компоненту нужно присвоить разницу между его исходным значением и 255:

```
// обрабатываем по 4 записи (1 px) за один раз
for (var i = 0; i < drawing.data.length; i = i + 4) {
    drawing.data[i] = 255 - drawing.data[i];     // инвертируем красный
    drawing.data[i+1] = 255 - drawing.data[i+1]; // инвертируем зеленый
    drawing.data[i+2] = 255 - drawing.data[i+2]; // инвертируем синий
}
```

После того как данные обработаны, нужно записать их обратно в элемент **canvas**, применив команду **putImageData(...)**:

```
mycontext.putImageData(drawing, 0, 0); // помещаем данные изображения
                                       // на "холст", начиная с точки (0,0)
```

Обсуждение

Как говорилось в рецепте 9.1, при визуализации на странице изображение превращается в битовые данные, наглядно представляющие пути и стили (или векторы), которые вы определили, программируя свой рисунок.

К счастью, элемент **canvas** позволяет обращаться к цветовым данным своих пикселов и менять их. Это означает, что вам доступны самые сложные

битовые трансформации. Помимо цветовых преобразований, которые мы рассмотрели выше, можно применять алгоритмы размывки, пикселизации, увеличения масштаба изображения и т. д. Подробное их рассмотрение не относится к тематике этой главы (и всей книги), однако они работают аналогично алгоритму, описанному в текущем рецепте: нужно получить массив данных растрового изображения, обработать записи цветовых компонентов (красного, зеленого, синего, альфа) для каждого пиксела и записать часть или все эти данные обратно в элемент **canvas**.

Если вы записываете на элемент canvas внешнее изображение из другого домена, не совпадающего с доменом размещения вашей веб-страницы, то «холст» помечается как «нечистый» (то есть не соблюдающий политику совпадения источников). В таком случае не получится извлечь данные с помощью getImageData(...).

Вы всегда можете записывать данные в элемент canvas, но считывать их разрешается, только если все данные «холста» относятся к тому же домену, которому принадлежит страница.

Дополнительная информация

В этом рецепте я лишь коснулся базовых вариантов манипулирования цветами изображений на «холсте». Намного более сложные эффекты, которые вы сможете применить к своим элементам **canvas**, обсуждаются на сайте PaintbrushJS: http://mezzoblue.github.com/PaintbrushJS/demo/.

9.7. Использование геометрических преобразований

Проблема

Необходимо применить к командам рисования какие-либо трансформации, например масштабирование, поворот, сдвиг и т. п.

Решение

В API **canvas** предусмотрено несколько команд, позволяющих преобразовать действия по рисованию на «холсте»:

- **translate(*x*, *y*)** — перемещение (транслирование) точки отсчета системы координат из **(0, 0)** в **(*x*, *y*)**;

- **scale(*x*, *y*)** — масштабирование единиц измерения системы координат по осям *x* и *y* независимо друг от друга;
- **rotate(*угол*)** — поворот системы координат вокруг точки отсчета **(0, 0)** на угол, заданный в радианах (по часовой стрелке).

Если вы решили комбинировать различные трансформации, то вам проще будет управлять элементом **canvas** через стек состояний. Прибегая к помощи команд управления стеком, вы всегда сможете вернуться на один уровень назад, одной командой отменив все преобразования (и другие изменения, например настройки стиля и цвета), выполненные с момента сохранения предыдущего состояния «холста». В API **canvas** есть две команды для управления стеком состояний: **save()** и **restore()**:

```
mycontext.save(); // сохраняем текущее состояние "холста"
mycontext.translate(10, 10); // перемещаем начало координат в точку (10,10)
mycontext.arc(0, 0, 10, 0, Math.PI * 2, true); // рисуем окружность
mycontext.stroke();
mycontext.restore();

mycontext.save();
mycontext.rotate(Math.PI / 4); // поворачиваем на 45° по часовой стрелке
mycontext.moveTo(0, 0);
mycontext.lineTo(10, 0);
mycontext.stroke();
mycontext.restore();
```

Все указанные действия несложно отменить: вернуть точку отсчета в исходное место командой **translate(-10, -10)** или восстановить исходный угол поворота командой **rotate(Math.PI / -2)**.

Однако если вы начнете объединять все возможные преобразования на разных уровнях, без стека состояний не обойтись. Во время работы со стилями или преобразованиями рекомендуется всегда управлять состояниями элемента **canvas** через стек.

Обсуждение

Наверное, идея о том, чтобы применять к командам рисования какие-либо трансформации, поначалу покажется вам несколько странной. Большинство разработчиков не сразу понимают ее. Но не беспокойтесь: поработав с этими командами несколько раз, вы привыкнете к новому стилю мышления, и такая концепция больше не будет казаться вам сложной.

Для начала расскажу, о чем вам необходимо очень хорошо помнить, применяя трансформации: в действительности они не влияют на команды рисования и текущее содержимое «холста». Преобразования меняют систему координат, то есть все координаты, которые вы укажете в последующих

командах рисования, будут автоматически интерпретироваться с учетом новой, а не исходной системы координат.

Непонятно? Ладно, попробую объяснить по-другому. Представьте, что вы удерживаете кончик пера в одной и той же точке пространства над листом бумаги и, не касаясь бумаги пером, перемещаете лист или поворачиваете его. Затем вы опускаете перо и рисуете так, как будто не двигали лист. После этого вы возвращаете лист бумаги в начальное положение на столе.

Еще раз прочитайте предыдущий абзац, чтобы хорошо понять, как трансформирование системы координат влияет на процесс рисования на «холсте».

Если вы транслируете (или переместите) систему координат на 10 пикселов вправо и начнете рисовать фигуру в точке (0, 0) относительно уже перемещенной системы, то в действительности эта фигура отобразится в точке (10, 0) абсолютной системы координат, то есть в точке (10, 0) *касаемо* содержимого элемента **canvas**.

Таким образом, транслирование системы координат на 10 пикселов вправо и рисование фигуры в точке (0, 0) даст такой же результат, как рисование фигуры в точке (10, 0) исходной системы координат, без дополнительного транслирования:

```
mycontext.save();
mycontext.translate(10, 0); // перемещаем систему координат
                            // на 10 пикселов правее
mycontext.moveTo(0, 0);
mycontext.lineTo(50, 0);    // в действительности линия появляется между
                            // точками (10,0) и (60,0)
mycontext.stroke();
mycontext.restore();

mycontext.moveTo(10, 0);
mycontext.lineTo(60, 0);    // фактически, та же линия
mycontext.stroke();
```

То же самое относится к повороту, только математика в этом случае чуть сложнее. Если повернуть «холст» на 30° по часовой стрелке и нарисовать фигуру в точке (50, 0), то вам покажется, что фигура находится на 30° ниже горизонта, так как координата точки будет трактоваться командой рисования относительно новой системы:

```
mycontext.save();
mycontext.rotate(Math.PI / 6); // поворачиваем на 30° по часовой стрелке
mycontext.moveTo(0, 0);
mycontext.lineTo(50, 0);       // фактически, линия наклонена на 30°
                               // относительно линии горизонта
mycontext.stroke();
mycontext.restore();
```

Поначалу вас может удивлять сама мысль о необходимости трансляции и поворота системы координат. В конце концов, почему нельзя просто нарисовать линию или окружность в правильном месте с правильной ориентацией и не забивать себе голову сложностями?

В некоторых задачах это возможно. Однако иногда возникают достаточно сложные задачи, в которых без преобразования системы координат попросту не получится грамотно и семантически верно записать команды рисования.

Как нарисовать наклонный прямоугольник? Конечно, можно рассчитать координаты вершин с помощью геометрических уравнений, а затем вручную нарисовать все четыре стороны как отдельные линии. Но кто захочет выполнять лишнюю работу? Вместо этого можно просто повернуть систему координат и вызвать команду **rect(...)** — и все будет гораздо проще.

Масштабирование — еще один пример операции, которая почти всегда требует преобразования системы координат, ведь масштабировать фигуры можно независимо по обеим осям x и y. Например, вы можете увеличить масштаб направления по оси x координатной системы в два раза, и тогда линия, для которой в коде задается длина 50 единиц (пикселов), будет визуализирована с коэффициентом 2, то есть длина видимого объекта будет равна 100 пикселам:

```
mycontext.save();
mycontext.scale(2, 1);   // масштабируем направление по оси x
                         // с коэффициентом 2
mycontext.moveTo(0, 0);
mycontext.lineTo(50, 0); // фактически, линия доходит от начала координат
                         // до точки (100,0)
mycontext.stroke();
mycontext.restore();

mycontext.moveTo(0, 0);
mycontext.lineTo(100, 0); // по сути, такая же линия
mycontext.stroke();
```

Другие задачи — скажем, поворот изображения — также подразумевают необходимость трансформирования, потому что крайне сложно (и совершенно нецелесообразно) вручную менять массив необработанных битовых данных. Вместо этого можно легко повернуть систему координат и нарисовать изображение на «холсте» обычным способом. Всю тяжелую работу за вас выполнит элемент **canvas**:

```
mycontext.save();
mycontext.rotate(Math.PI / 4);  // поворачиваем на 45° по часовой стрелке
mycontext.drawImage(img, 0, 0); // рисуем изображение в точке (0,0)
                                // повернутой системы координат
mycontext.restore();
```

Наконец, взгляните на вложенные преобразования, чтобы в очередной раз убедиться в удобстве и полезности управления состояниями «холста» через стек (результат выполнения следующего кода показан на рис. 9.12):

```
mycontext.beginPath();
mycontext.strokeStyle = "#f00"; // красный цвет
mycontext.translate(20, 20);    // перемещаем точку отсчета системы координат
                                // в точку (20,20)
mycontext.moveTo(0, 0);         // на самом деле это (20,20)
mycontext.lineTo(80, 10);       // на самом деле это (100,30)
mycontext.stroke();

mycontext.save(); // сохраняем состояние элемента <canvas>

mycontext.beginPath();
mycontext.strokeStyle = "#00f"; // теперь синий цвет
mycontext.rotate(Math.PI / 4);
mycontext.moveTo(0, 0);
mycontext.arc(0, 0, 52, Math.PI / 3, Math.PI / 6, true);
mycontext.closePath();    // соединяем конец пути с начальной точкой
mycontext.stroke();

mycontext.restore();      // возвращаем предыдущее состояние элемента <canvas>

mycontext.beginPath();
mycontext.moveTo(80, 10);
mycontext.lineTo(14, 50);
mycontext.stroke();
```

Рис. 9.12. Преобразование системы координат и управление стеком состояний «холста»

Дополнительная информация

Обратитесь к спецификации трансформаций «холста» на сайте W3C: http://www.w3.org/TR/2dcontext/#transformations.

9.8. Добавление на «холст» текста

Проблема

Необходимо поместить текст поверх рисунка на «холсте».

Решение

В API **canvas** предусмотрено две команды для визуализации текста на «холсте»: **fillText(...)** и **strokeText(...)**. Обе команды принимают один набор параметров: (***строка_для_визуализации***, ***x***, ***y***[, ***максимальная_ширина***]). Единственная разница между ними заключается в наличии у текста заливки и обводки или только обводки.

Для установки стиля шрифта (гарнитура, размер и т. п.) используется свойство **font**:

```
mycontext.font = "25pt Arial";
```

После этого нужно вызвать подходящую команду работы с текстом:

```
mycontext.fillText("Hello World", 0, 25);
mycontext.strokeText("Hello World", 0, 75);
```

Результат выполнения этого фрагмента кода показан на рис. 9.13.

Обсуждение

Команды **fillText(...)** и **strokeText(...)** применяют для визуализации текста стиль, установленный свойством **font**. Доступные параметры (гарнитура шрифта, стиль, размер и т. п.) — такие же, как и для правил определения стилей шрифта в CSS. Для настройки цветов аналогично используются свойства **fillStyle** и **strokeStyle** соответственно.

Дополнительная информация

Подробнее о выводе текста на «холст» рассказывается на сайте https://developer.mozilla.org/en/drawing_text_using_a_canvas.

Рис. 9.13. Текст с заливкой и текст с обводкой

9.9. Обрезка рисунков на «холсте»

Проблема

Вы собираетесь применить команду рисования, но хотели бы обрезать рисунок по определенному контуру.

Решение

В API **canvas** есть команда **clip(...)**, которая принимает в качестве параметра путь, определенный предыдущими командами рисования, и использует его как *маску обрезки* для последующих команд. Это означает, что изображение на элементе **canvas** появляется только внутри контура этой маски обрезки, а все лежащее за его пределами отбрасывается.

Следующий фрагмент кода позволяет нарисовать на «холсте» букву «Н», обрезав ее маской в форме окружности:

```
mycontext.beginPath();
mycontext.arc(50, 50, 25, 0, Math.PI * 2, true); // путь в форме окружности
mycontext.clip(); // превращаем путь в маску обрезки
```

```
mycontext.fillStyle = "#f00";
mycontext.font = "50pt Arial";
mycontext.fillText("H", 25, 75);
```

Результат — обрезанная по краям буква «H» — показан на рис. 9.14.

Рис. 9.14. Пример обрезания буквы контуром окружности

Как видите, сама окружность не визуализируется, а лишь используется как контур обрезки для последующей команды рисования **fillText**.

Обсуждение

По умолчанию состояние элемента **canvas** таково, что его маска обрезки включает всю видимую область «холста». Когда вы определяете маску обрезки в команде **clip(...)**, она действует на все последующие команды рисования до тех пор, пока вы не измените или не сбросите маску.

Аналогично тому, что вы видели в рецепте 9.7, стек состояний элемента позволяет создать временную маску обрезки, а затем без труда восстановить первоначальную маску, которая охватывает весь элемент (результат выполнения следующего кода показан на рис. 9.15):

```
mycontext.save();
mycontext.beginPath();
```

```
mycontext.arc(50, 50, 25, 0, Math.PI * 2, true); // путь в форме окружности
mycontext.clip();    // превращаем путь в маску обрезки

mycontext.fillStyle = "#f00";
mycontext.font = "50pt Arial";
mycontext.fillText("H", 25, 75);
mycontext.restore(); // возвращаем состояние элемента <canvas> по умолчанию
                     // (в том числе стандартное состояние маски обрезки)

mycontext.font = "25pt Arial";
mycontext.fillText("ello World", 70, 70); // черный текст без обрезки
```

Рис. 9.15. Восстановление предыдущего состояния маски обрезки

Дополнительная информация

Обратитесь к спецификации маски обрезки на сайте W3C: http://www.w3.org/TR/2dcontext/#clipping-region.

9.10. Добавление эффекта анимации к рисункам на «холсте»

Проблема

Интересно рисовать на «холсте» статические фигуры, но вам бы хотелось, чтобы графические объекты могли двигаться.

Решение

Что касается анимации на «холсте», то все сводится к простой замене изображений: вы рисуете один кадр, через несколько миллисекунд стираете его и рисуете следующий кадр, на котором отдельные элементы чуть-чуть сдвинулись или поменялись иным образом. Если смена кадров происходит достаточно быстро — 20–30 кадров в секунду, — то создается впечатление качественной анимации.

Мы уже обсудили в этой главе рисование различных объектов на элементе **canvas**. Кроме того, мы упомянули один из способов очистки «холста» (путем переопределения его ширины). Для создания анимированного изображения нужно соединить обе концепции.

Рассмотрим простейший пример создания движущегося красного круга (результат выполнения кода показан на рис. 9.16):

```
function draw_circle(x, y) {
    mycontext.fillStyle = "#f00";
    mycontext.beginPath();
    mycontext.arc(x, y, 10, 0, Math.PI * 2, true);
    mycontext.fill();
}

function erase_frame() {
    mycanvas.width = mycanvas.width;
}

var ball_x = 50;
var ball_y = 50;
var delta = 3;

draw_circle(ball_x, ball_y);

setInterval(function(){
    if (ball_x > 100 || ball_y < 15 || ball_x < 15 || ball_y > 100) {
        delta *= -1;
    }
    ball_x += delta;
    ball_y += delta;
    erase_frame();
    draw_circle(ball_x, ball_y);
}, 35);
```

Обсуждение

В зависимости от композиции кадров, для эффективного рисования (и стирания) фигур и/или изображений следует применять разные техники. Рассмотрим несколько подходов к рисованию и стиранию объектов для успешной имитации анимации.

Если вы работаете с прозрачным фоном, на котором двигается одна фигура (например, круг), как в примере выше, то вам не нужно стирать все содержимое элемента **canvas** перед рисованием следующего кадра. Достаточно очистить небольшую область «холста», на которой в данный момент что-то нарисовано.

Рис. 9.16. Движущееся изображение красного круга на «холсте»

Таким образом, перед вами стоит задача очистить ту область, на которой вы нарисовали предыдущий кадр. Техника частичной очистки может значительно повысить производительность, особенно если размеры «холста» достаточно велики.

Для очистки части «холста» используйте команду **clearRect(...)**:

```
function draw_circle(x, y) {
    mycontext.fillStyle = "#f00";
    mycontext.beginPath();
    mycontext.arc(x, y, 10, 0, Math.PI * 2, true);
    mycontext.fill();
}

function erase_circle(x, y) {
    mycontext.clearRect(x-10, y-10, 20, 20);
}

var ball_x = 50;
var ball_y = 50;
var delta = 3;
```

```
draw_circle(ball_x, ball_y);

setInterval(function(){
    if (ball_x > 100 || ball_y < 15 || ball_x < 15 || ball_y > 100) {
        delta *= -1;
    }
    erase_circle(ball_x, ball_y);
    ball_x += delta;
    ball_y += delta;
    draw_circle(ball_x, ball_y);
}, 35);
```

Техника, подразумевающая перерисовывание кадра целиком, работает прекрасно, но в определенных ситуациях недостаточно эффективно. Например, вы создаете анимацию, где некая фигура вроде нашего красного круга двигается поверх статической картинки (или изображения, фотографии и т. п.). Стирание и перерисовывание статического фонового изображения 30 раз в секунду только потому, что фигура на переднем плане должна перемещаться, — это излишняя трата ресурсов.

Одно из решений такой проблемы — использование двух элементов **canvas**, наложенных один на другой. На фоновом элементе **canvas** вы рисуете статическую сцену, а на верхнем создаете анимацию красного круга, как рассказано выше. Таким образом, фоновая картинка визуализируется только однажды, а не при каждом перерисовывании слоя с движущимся изображением (рис. 9.17).

Рис. 9.17. Фоновый «холст» содержит статическое изображение; сверху на него наложен второй холст с анимированным красным кругом

Возможно, перерисовывать целый «холст» только для того, чтобы подвигать по нему красный круг, — это слишком. Однако на практике часто приходится анимировать не только положение объекта. Например, объект можно двигать и одновременно поворачивать. В таком случае подход со стиранием и перерисовыванием оказывается наиболее эффективным.

К решению нашей простой задачи можно подойти с другой стороны: создать на переднем плане элемент **canvas** такого размера, чтобы на нем уместился только красный круг и больше ничего (20×20 пикселов), а затем двигать по экрану сам «холст», применяя для этого команды позиционирования CSS. В данной ситуации вынос красного круга на отдельный «холст» поверх фонового изображения особенно удобен, так как два элемента **canvas** можно будет размещать на экране независимо друг от друга.

Дополнительная информация

Простейшее руководство по анимации на «холсте» вы найдете на сайте https://developer.mozilla.org/en/Canvas_tutorial/Basic_animations.

9.11. Рисование графиков на «холсте»

Проблема

У вас есть данные, и вы хотели бы построить на их основании график, используя элемент **canvas**.

Решение

Все команды API **canvas**, перечисленные в этой главе, а также другие усложненные команды (для создания сложных кривых и т. п.) в сочетании друг с другом позволяют создавать изображения для визуализации данных: линейные графики, круговые диаграммы и графики других типов.

Вы, конечно, понимаете, что создание графика — довольно сложная задача, поэтому мы не станем подробно рассматривать рисование графиков с нуля. Вместо этого я продемонстрирую вам, как с успехом применять простую и бесплатную библиотеку под названием flot (хотя это не означает, что вам не придется основательно потрудиться!).

Библиотека flot построена на базе jQuery, поэтому сначала нужно загрузить новейшую версию jQuery, а затем последний выпуск библиотеки flot:

```
<script src="jquery.js"></script>
<script src="jquery.flot.js"></script>
```

После этого вы должны создать на странице элемент-заполнитель, в который flot сможет поместить «холст» с графиком. Библиотека автоматически подгоняет пиксельные размеры создаваемого элемента **canvas** под размеры элемента-заполнителя:

```
<div id="my_graph" style="width:600px; height:300px"></div>
```

На следующем этапе необходимо подготовить данные, которые будут выведены на график. Можно либо загрузить их динамически с помощью Ajax, либо вставить прямо в код сценария, как показано здесь:

```
var graph_data = [[0, 3], [4, 8], [8, 5], [9, 13]];
```

Что касается формата данных, то это массив, состоящий из пар **[*X*, *Y*]**. Одна пара описывает одну точку данных и включает значения по оси x и по оси y соответственно.

Итак, у вас есть данные для графика. Если его настройки по умолчанию вас удовлетворяют, просто вызовите API библиотеки flot и передайте ему свои данные:

```
var my_graph = $("#my_graph");  // получаем ссылку на элемент-заполнитель
$.plot(my_graph, [graph_data]); // передаем данные графика
                                // в виде одного ряда
```

Обратите внимание, что данные графика переданы в квадратных скобках. Это означает, что передан массив данных. API построения графика библиотеки flot поддерживает вывод нескольких последовательностей данных одновременно на одном графике. В примере передается только одна последовательность, однако вы могли бы при необходимости добавить и другие последовательности данных. Настройки и стили по умолчанию позволяют получить симпатичный график (рис. 9.18).

Как видите, всего несколько строк кода позволили эффективно применить возможности библиотеки flot и создать качественный график, и не пришлось самостоятельно писать сотни строк и обращаться к командам API **canvas**.

Если хотите поупражняться, то попробуйте применить полученные в этой главе знания об API «холста» и проверить, сможете ли вы воспроизвести аналогичный график, не прибегая к помощи библиотеки flot.

API библиотеки flot позволяет вручную генерировать и выводить метки для осей x и y, задавать минимальный и максимальный диапазон отображаемых

данных для каждой оси, вручную определять размер шага сетки для оси, настраивать цвета и стили линий. Кроме того, библиотека flot поддерживает множество других возможностей.

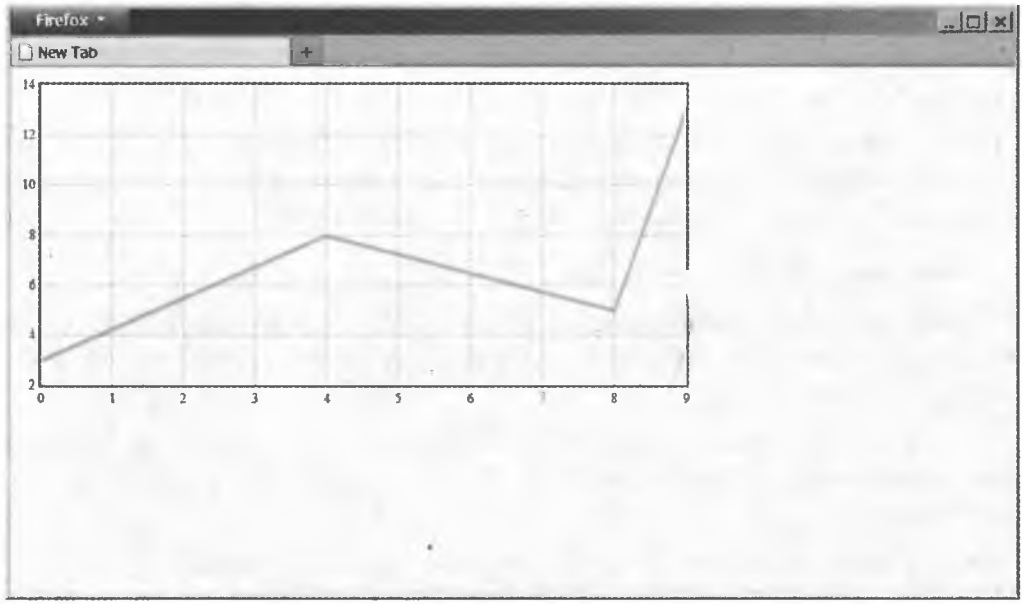

Рис. 9.18. Построение на элементе <canvas> графика по точкам данных с помощью библиотеки flot

Обсуждение

Библиотека flot отличается поразительной гибкостью и мощью. Она предлагает целый диапазон параметров для управления деталями построения графиков. Мы не будем в этой главе рассматривать API flot во всех подробностях, но я настоятельно рекомендую вам самостоятельно изучить его и ознакомиться с параметрами конфигурации по адресу http://people.iola.dk/olau/flot/API.txt.

Помимо встроенных параметров построения графиков, flot поддерживает систему подключаемых модулей, позволяющую другим разработчикам расширять возможности библиотеки. Благодаря этому flot становится чрезвычайно мощным решением для визуализации графиков в браузерах, которое можно применять на любых сайтах. В дистрибутив flot входят несколько полезных модулей, а множество других доступны для загрузки и могут применяться при необходимости.

Разумеется, существуют десятки других библиотек для построения графиков на «холсте» — одни бесплатные, другие требуют покупки лицензии. Они обладают огромным количеством разнообразных возможностей, так что не теряйтесь, исследуйте, и вы обязательно найдете то, что подходит вам лучше всего.

Дополнительная информация

Подробнее о том, как с помощью flot строить графики на элементе **canvas**, рассказывается на домашней странице библиотеки: http://code.google.com/p/flot.

9.12. Сохранение рисунка с «холста» в файле

Проблема

Необходимо сохранить текущее содержимое элемента **canvas** в отдельном файле.

Решение

В API **canvas** есть команда **toDataURL(...)**, умеющая извлекать данные изображения из элемента **canvas** и преобразовывать их в определенный формат. Например, для извлечения данных графика, построенного с помощью библиотеки flot (см. рецепт 9.11), в формате PNG нужно применить такой код:

```
var canvas = $("canvas", my_graph)[0];
var image_data = canvas.toDataURL("image/png");
```

Получив необработанные данные изображения, просто сохраните их в файл.

Для отправки данных на сервер с помощью jQuery через Ajax можно использовать такую команду:

```
$.post("http://location/path/to/upload", {data: image_data});
```

Обсуждение

Извлечь необработанные данные изображения из элемента легко, сложнее фактически сохранить их в файл.

Чтобы сохранить файл на сервере, нужно лишь отправить туда данные в виде строкового значения с помощью Ajax. По прибытии данных на серверную

сторону вам необходимо разобрать формат dataURI, декодировать данные изображения (из формата base64) и сохранить полученную информацию в файл.

Если вы хотите разрешить пользователю сохранить информацию на компьютере, то прикажите браузеру загрузить только что созданный файл с сервера и предложить пользователю записать его на жесткий диск.

Некоторые свежие версии браузеров поддерживают локальный доступ к файлам, так что теоретически пользователь может сохранить изображение на своем компьютере без необходимости сначала загружать его на сервер, а потом скачивать оттуда.

Поскольку процесс обработки таких файлов довольно сложен (и не относится к тематике книги), я рекомендую вам прибегнуть к помощи инструмента Canvas2Image, который сделает за вас значительную часть утомительной работы (http://www.nihilogic.dk/labs/canvas2image/).

Дополнительная информация

Больше информации о сохранении изображений с «холста» на сервер (с помощью PHP) и предоставлении их пользователю для загрузки можно найти по адресу http://blog.kevinsookocheff.com/saving-canvas-data-to-an-image-file-with-java-61171.

10 Расширенные возможности JavaScript в HTML5

Кайл Симпсон

10.0. Введение

По сравнению с предыдущими версиями языка, область применения HTML5 значительно расширилась. HTML4 в основном фокусировался на разметке, а особенностью HTML5 в числе прочего можно назвать множество сложных API JavaScript, позволяющих реализовывать потрясающие возможности. Большинство новых технологий описываются в собственных спецификациях и создаются другими рабочими группами, поэтому не совсем корректно объединять весь набор технологий под общим названием «HTML5». По этой причине — чтобы не сбивать никого с толку — подобные API называют *компаньонами* HTML5, а кто-то даже придумал забавное выражение «HTML5 и его друзья». Итак, в этой главе вы познакомитесь с некоторыми друзьями HTML5.

Мы уже касались темы API JavaScript, когда рассматривали элементы, зависящие от привязки к макету, например **canvas**, **audio** и **video**. Здесь же мы сосредоточимся на других компаньонах API, не обязательно относящихся к конкретному элементу разметки. Воспринимайте рецепты этой главы как дополнение к тому, что необходимо для создания по-настоящему интерактивного веб-приложения.

Степень стандартизации и поддержки технологий, которые мы будем обсуждать в этой главе, может сильно отличаться. Они по-разному реализованы в существующих браузерах, так как среди создателей браузеров нет четких договоренностей относительно этих технологий. Таким образом,

используя их, необходимо тщательно планировать свои действия и соблюдать осторожность. Одни технологии возвращают к более старой и бедной функциональности сайтов в неподдерживающих их браузерах; в других же случаях приходится встраивать в приложения различные обходные пути для имитации функций, которые отсутствуют или недоступны.

Поскольку развитие технологий идет чрезвычайно быстрыми темпами, и к моменту прочтения этой книги некоторые из них могут уже устареть, я не указываю точные названия и версии браузеров, поддерживающих те или иные функции. Тестируйте решения в браузерах, популярных у вашей целевой аудитории, и у вас всегда будет самая свежая информация о степени поддержки нужной функциональности в интересующих вас средах.

10.1. Локальное хранилище

Проблема

Вам необходимо сохранить какие-то данные (например, пользовательские настройки или информацию из частично заполненной формы) на компьютере пользователя, чтобы к ним можно было обратиться при его следующем визите на ваш сайт.

Решение

В HTML5 появилось два новых контейнера для постоянного хранения данных в браузере: **sessionStorage**, который сохраняет данные до окончания сеанса пользователя или до закрытия экземпляра браузера, и **localStorage**, сохраняющий данные «навсегда» (что означает «до тех пор, пока либо код, либо сам пользователь не удалит их»).

К обоим контейнерам можно обращаться через один и тот же API. Основное различие между ними заключается в том, как долго браузер хранит данные.

Данные, сохраняемые в этих контейнерах, должны относиться к строковому типу. Если возникнет задача записать в хранилище сложные объекты, то одним из вариантов решения может стать их сериализация в JSON с помощью функции JSON.stringify().

Если нужно проверить, поддерживает ли браузер эти API, воспользуйтесь следующей строкой:

```
var storage_support = window.sessionStorage || window.localStorage;
```

Для того чтобы сохранить данные на время работы браузера (то есть обеспечить их удаление в тот момент, когда пользователь закроет окно браузера), используйте **sessionStorage**:

```
var user_id = "A1B2C3D4";
var user_data = {
    name: "Tom Hanks",
    occupation: "Actor",
    favorite_color: "Blue"
    // ...
};

sessionStorage.setItem(user_id, JSON.stringify(user_data));
```

Чтобы сохранить данные на более длительный период времени, применяйте **localStorage**:

```
var user_id = "A1B2C3D4";
var user_prefs = {
    keep_me_logged_in: true,
    start_page: "daily news"
    // ...
};

localStorage.setItem(user_id, JSON.stringify(user_prefs));
```

Эти фрагменты данных выглядят почти идентичными именно потому, что у них одинаковые API.

Для извлечения данных из контейнера хранилища (разумеется, если они там есть) используйте код, подобный следующему:

```
var user_id = "A1B2C3D4";
var user_data = { /* defaults */ };
var user_prefs = { /* defaults */ };

if (sessionStorage.getItem(user_id)) {
    user_data = JSON.parse(sessionStorage.getItem(user_id));
}
if (localStorage.getItem(user_id)) {
    user_prefs = JSON.parse(localStorage.getItem(user_id));
}
```

Эти API позволяют с легкостью сохранять и извлекать такие данные, как пары из ключей и значений, где последние относятся к строковому типу. При этом важно понимать, что подобная строка может представлять собой все, что угодно, включая строковую сериализацию сложного объекта данных.

Контейнеры localStorage и sessionStorage реализованы по принципу синхронного выполнения, что упрощает их использование, однако может привести к снижению производительности. Будьте очень аккуратны, применяя данные из этих API в коде, где важны показатели производительности.

Обсуждение

Когда речь заходит о сохранении данных на клиентской стороне, разработчику наверняка сразу вспоминаются файлы cookie. Однако с ними связано несколько проблем, делающих cookie далеко не лучшим решением для хранения пользовательских данных. В этой главе мы рассмотрим новейшую альтернативу: API хранения HTML5 (также известные как «хранилища DOM»).

Во многих браузерах данные в контейнере sessionStorage сохраняются даже после сбоя. Таким образом, применение sessionStorage — прекрасное решение для временного зеркалирования информации, вводимой пользователем в поля формы: если произойдет сбой браузера, вы сможете восстановить то, что пользователь успел написать.

Рассмотрим спецификацию API **sessionStorage** и **localStorage**.

- **getItem(*ключ*)** — метод возвращает из контейнера для хранения элемент данных, соответствующий указанному ключу.
- **setItem(*ключ*, *значение*)** — этот метод добавляет в контейнер элемент данных с указанным ключом и значением.
- **key(*индекс*)** — функция возвращает ключ элемента данных с указанным числовым индексом.
- **removeItem(*ключ*)** — метод удаляет из контейнера для хранения элемент данных с указанным ключом.
- **clear()** — метод очищает текущий контейнер, то есть удаляет из него все данные.
- **length** — свойство указывает, сколько элементов данных находится в контейнере.

Обычно браузеры выделяют для хранилищ до 5 Мбайт дискового пространства. Этого более чем достаточно для большинства создаваемых приложений. Однако будьте готовы к тому, что вам придется распознавать и обрабатывать ошибки, если вы будете пытаться записывать больше данных, чем позволяет браузер.

В отличие от файлов cookie (для которых явно определяется срок действия) и контейнера **sessionStorage** (для которого срок действия определяется неявно и завершается с закрытием экземпляра браузера или окончанием сессии), данные API **localStorage** никогда не удаляются автоматически. У этой особенности есть как плюсы, так и минусы.

Преимущество состоит в том, что данные остаются на жестком диске так долго, как вам это требуется, либо до тех пор, пока пользователь самостоятельно не очистит хранилище. Однако при таком подходе достигнуть лимита

в 5 Мбайт можно быстрее, чем ожидалось, особенно если о старых данных никто не вспоминает, и они навсегда остаются в хранилище.

Одно из наиболее распространенных решений — реализация собственного механизма проверки срока действия. В этом случае для каждого элемента данных сохраняется отметка времени, которая затем вручную проверяется при любой загрузке страницы. После проверки старые данные при необходимости удаляются. Например, ваш интернет-магазин хранит историю всех продуктов, просмотренных пользователем во время многочисленных посещений, и отображает эту информацию в разделе наподобие «Вы смотрели раньше». Однако вы не хотите, чтобы приложение запоминало эту информацию навсегда, поэтому можете вручную задать срок хранения и удалять элементы, например, созданные более 21 дня назад:

```
// во-первых, сохраняем просматриваемый в данный момент
// товар в истории
var current_item = {
    id: "ABCD0123",
    data: "Mens' Running Shoes",
    ts: new Date()   // отметка времени, будет использоваться
                     // для проверки срока хранения элемента
};
localStorage.setItem(current_item.id, JSON.stringify(current_item));

// теперь вручную удаляем "просроченные" старые записи
var key, data;

for (var i=0; i<localStorage.length; i++) {
    key = localStorage.key(i);
    data = localStorage.getItem(key);
    if (data.ts < ((new Date()) - 60*60*24*21)) { // старше 21 дня
        localStorage.removeItem(key);
    }
}
```

Оба API ограничивают доступ на чтение и запись данных тем доменом (поддоменом, схемой, портом и т. д.), на котором размещена веб-страница, то есть невозможно из сценария на какой-то странице обратиться к данным, находящимся в другом домене. С одной стороны, это полезно, но с другой стороны, ограничивает функциональность. Несмотря на то что данные хорошо защищаются (например, от сайтов-шпионов), различные службы одного приложения, находящиеся в разных доменах, не смогут совместно работать через эти интерфейсы с одним набором данных.

Дополнительная информация

Подробнее об API хранилищ DOM рассказывается на сайте MDC: https://developer.mozilla.org/en/dom/storage.

10.2. Кэширование приложений

Проблема

Необходимо обеспечить пользователям доступ к веб-приложению (и всем его ресурсам) даже в автономном режиме, но без подключения стандартного кэша браузера.

Решение

В HTML5 определяется специальный кэш приложений — **appcache**. С его помощью вы можете приказать браузеру сохранить определенные ресурсы: изображения, код CSS или JS и т. п., чтобы они были доступны приложению, даже если пользователь запустит браузер в автономном режиме, без подключения к Интернету. Чтобы проверить, поддерживает ли браузер функциональность **appcache**, используйте такую строку:

```
var appcache_support = !!window.applicationCache;
```

Чтобы применить **appcache** в своем приложении, сначала необходимо создать файл манифеста с перечислением всех ресурсов, которые должны попасть в кэш. Такой файл выглядит следующим образом:

```
CACHE MANIFEST

CACHE:
index.html
help.html
style/default.css
images/logo.png
images/backgound.png
```

Файл манифеста должен включать раздел **CACHE**, где перечисляются ресурсы, предназначенные для сохранения в **appcache**. Можно также добавить раздел **NETWORK** и указать там URL-адреса, которые должны вызываться динамически (например, через Ajax), без сохранения в кэше. Еще один необязательный раздел называется **FALLBACK** и содержит список локальных (кэшированных) «альтернативных» файлов, которые подгружаются при невозможности выполнить удаленные URL-запросы (например, это может быть контент по умолчанию для автономного сценария, замещающего обычный сетевой вызов серверного API). Итак, манифест кэша создан — в примере мы назвали его **cache.manifest**. Теперь расскажите браузеру об этом файле, добавив в элементе **html** страницы новое свойство:

```
<html manifest="cache.manifest">
```

Приложениям, выполняющимся в контексте appcache, запрещается соединяться с удаленными URL-адресами, за исключением URL-адресов, перечисленных в разделе NETWORK манифеста кэша. Это обеспечивает дополнительный уровень безопасности приложений, так как они не могут обращаться ни к каким серверам, кроме известных и внесенных в белый список.

Вот и все! Сейчас ваше приложение может использовать **appcache** и работать автономно. Браузер будет постоянно кэшировать перечисленные ресурсы и не станет запрашивать их повторно при обновлении страницы, пока файл манифеста кэша не изменится.

Обновление кэша приложений. *Недостаточно* лишь поменять файл, копия которого сохранена в пользовательском кэше приложений, чтобы новая версия автоматически загрузилась на компьютер пользователя. Браузер проверяет наличие новых версий файлов в **appcache** (и загружает измененные файлы) только в том случае, если меняется файл манифеста.

Файл манифеста кэша должен обслуживаться с использованием MIME-типа text/cache-manifest.

Однако в реальной жизни вполне возможна такая ситуация, когда нет ни новых ресурсов, которые следует добавить в файл манифеста, ни необходимости удалить из списка какие-то существующие страницы. В таком случае вам, по сути, не требуется менять файл манифеста.

Самый простой выход — добавить в файл комментарий, который можно будет обновлять, указывая новое значение каждый раз, когда один или несколько ресурсных файлов поменяются, и придется заставить браузер обновить их в **appcache**:

```
CACHE MANIFEST
# cache version: 1257

CACHE:
index.html
help.html
style/default.css
images/logo.png
images/background.png
```

После каждого обновления любого из ресурсов, которые кэшируются в **appcache**, просто увеличивайте на единицу номер версии в файле манифеста. Когда пользователь в очередной раз загрузит страницу, браузер увидит, что файл манифеста изменился, и обновит содержимое **appcache**.

Однако существует одна опасность. Даже если браузер во время загрузки страницы заметит, что файл манифеста обновился, он не станет сразу прерывать загрузку текущей страницы, а дождется ее завершения. Таким образом, на экране появится устаревшее содержимое кэша приложений. Только после этого браузер в фоновом режиме запросит обновленные файлы, чтобы подготовить их для *следующей* загрузки страницы.

К счастью, на помощь приходит один из инструментов JavaScript: интерфейс **applicationCache**. С его помощью вы можете убедиться, что было загружено и теперь доступно новое содержимое **appcache**, а затем принудительно поместить его в актуальный кэш приложений, не дожидаясь обновления страницы:

```
var cache = applicationCache;
cache.addEventListener("updateready", function(){
    if (cache.stats == cache.UPDATEREADY) {
        cache.swapCache(); // подгрузить в кэш новые элементы
    }
}, false);
```

Благодаря этому новые элементы кэша приложений становятся доступными для использования в течение текущего сеанса страницы.

Однако такие места на странице, где кэшированные ресурсы уже были применены, не обновляются. В зависимости от ситуации это может привести к различным странным последствиям.

Самый очевидный способ обновления всего контента страницы с учетом новых версий ресурсов — повторная загрузка. Однако нельзя забывать, что со страницей работает пользователь, и мы не должны мешать ему. Поэтому перед повторной загрузкой страницы стоит спросить у него разрешения:

```
var cache = applicationCache;
cache.addEventListener("updateready", function(){
    if (cache.stats == cache.UPDATEREADY) {
        if (confirm("Этот сайт был обновлен. Хотите перезагрузить страницу?")) {
            location.reload();
        }
    }
}, false);
```

До сих пор мы рассматривали только случаи, когда файл **appcache** помечается как требующий обновления при очередной загрузке страницы. Однако некоторые сайты специально разрабатываются так, чтобы подолгу оставаться открытыми в браузере пользователя и не требовать или почти не требовать перезагрузки страниц. В таком случае процесс получения обновлений подразумевает специальную обработку.

Чтобы заставить браузер проверить, не был ли обновлен файл **cache.manifest**, и при необходимости загрузить новое содержимое **appcache**, следует вызвать метод API **update()**:

```
function updateAppcache(){
    var cache = applicationCache;
    cache.update(); // проверяем, не обновлялся ли файл манифеста кэша

    cache.addEventListener("updateready", function(){
        if (cache.stats == cache.UPDATEREADY) {
            if (confirm("Этот сайт был обновлен. Хотите перезагрузить страницу?"))
{
                location.reload();
            }
        }
    }, false);
}
```

Чаще всего создают такие решения, когда проверка автоматически выполняется либо через определенные интервалы времени, скажем, каждые 24 часа, либо в ответ на действия пользователя, например нажатие кнопки на странице.

Если вы решите обновлять кэш приложений автоматически, через определенные интервалы времени, то стоит проявлять заботу и предупреждать пользователя, прежде чем страница перезагрузится. Если же пользователь сам будет инициировать обновления, то можно безопасно перезагружать страницу без предупреждения, как только станет известно об обновлении кэша приложений.

Обсуждение

Мы уже упоминали, что кэш **appcache** — особенная технология. Причина уникальности заключается в том, что, в отличие от содержимого кэша браузера, у ресурсов, хранящихся в **appcache**, нет «срока годности» (здесь обнаруживается его сходство с интерфейсом **localStorage**, о котором мы говорили в рецепте 10.1).

На практике это означает, что любые ресурсы, перечисленные в манифесте, кэшируются навсегда. В частности, браузер не запрашивает их повторно при последующих обновлениях страницы.

Непрерывное кэширование ресурсов повышает производительность повторных загрузок страницы, особенно на мобильных устройствах, подключенных к каналам с ограниченной пропускной способностью. Однако иногда возникает необходимость в обновлении одного или нескольких элементов постоянного кэша.

Заставить браузер очистить кэш и обновить перечисленные в манифесте ресурсы можно двумя способами: первый подготавливает страницу к очередному обновлению, а второй применяется для обновления контента во время одного длительного сеанса работы со страницей.

Если для страницы уже определен кэш приложений, то браузер использует для ее визуализации ресурсы из существующего кэша, даже если в момент загрузки окажется, что файл манифеста был обновлен и содержимое **appcache** устарело. Любые обновления содержимого **appcache** становятся доступными только при очередной перезагрузке страницы.

Добавляя в код JavaScript событие **UPDATEREADY**, как показано выше, вы можете определять, не устарели ли кэшированные элементы, благодаря которым визуализировалась страница, и при необходимости принудительно обновлять **appcache**, не дожидаясь перезагрузки страницы пользователем. Таким образом, последующие обращения к ресурсам в течение текущего сеанса работы будут возвращать уже обновленные версии файлов. Кроме того, если пользователь не возражает, вы можете полностью обновить страницу, чтобы подгрузить все новые элементы **appcache**.

Считается, что **appcache** надежнее кэширует автономные ресурсы приложений, чем обычный кэш браузера. Помимо этого, он предоставляет JavaScript API для поддержки обновлений — а такая функциональность недоступна при использовании обычного кэша браузера.

Дополнительная информация

В руководстве для новичков вы сможете узнать основы функциональности **applicationCache**: http://www.html5rocks.com/en/tutorials/appcache/beginner/. Более подробное описание API **applicationCache** вы найдете в следующей записи на сайте MDC: https://developer.mozilla.org/en/offline_resources_in_firefox. Прекрасное обсуждение поведения **appcache** можно найти по адресу http://appcachefacts.info.

10.3. Перетаскивание

Проблема

Необходимо реализовать «родную» функциональность перетаскивания, не обрабатывая события мыши вручную и не прибегая к помощи сложной библиотеки.

Решение

Понимая, насколько важна и распространена функциональность перетаскивания (drag-and-drop) в современных сложных веб-приложениях, разработчики HTML5 предусмотрели прямой API для обработки этих действий.

Чтобы узнать, поддерживается ли в браузере «родная» функциональность перетаскивания, можно использовать следующую строку кода:

```
var dnd_support = 'draggable' in document.createElement('span');
```

Теперь создадим простую программу, демонстрирующую функциональность перетаскивания. Начнем с установки визуальных стилей для элементов:

```
<style>
#foobar { background-color:yellow; width:100px; height:100px; cursor:move; }
#catcher { background-color:blue; width:150px; height:150px; padding:5px;
    margin-bottom:5px; }
</style>
```

На первом этапе нужно определить атрибут **draggable** элемента, который мы хотим сделать перетаскиваемым:

```
<div id="catcher">...</div>
<div id="foobar" draggable="true">...</div>
```

Теперь с помощью API JavaScript и событий перетаскивания необходимо сообщить браузеру, в какую область можно перетаскивать элемент и что с ним сделать, когда пользователь отпустит его над указанной областью.

Например, можно перехватывать событие **dragstart** и применять к элементу другой стиль на то время, пока выполняется перетаскивание (скажем, создавать вокруг него рамку или делать его частично прозрачным):

```
var foobar = document.getElementById("foobar");
foobar.addEventListener("dragstart", function(evt) {
    this.style.border = "3px dotted #000"; // черная пунктирная рамка
}, false);
```

Теперь определим стиль «зоны приема», чтобы было очевидно, что именно здесь, а не в любой другой части экрана можно оставить перетаскиваемый элемент:

```
var catcher = document.getElementById("catcher"); // подхватываем
                                                  // перемещенный элемент
catcher.addEventListener("dragenter", function(evt) {
    this.style.border = "3px solid red";          // создаем для зоны приема
                                                  // красную рамку
}, false);

catcher.addEventListener("dragleave", function(evt) {
    this.style.border = ""; // удаляем рамку зоны приема
}, false);

catcher.addEventListener("dragover", function(evt) {
    if (evt.preventDefault) evt.preventDefault();
    return false;
}, false);
```

В предыдущем фрагменте кода мы добавили к элементу, получающему перетаскиваемый элемент, приемники событий, которые перехватывают события **dragover**, **dragenter** и **dragleave**. События **dragenter** и **dragleave** лишь включают и выключают отображение красной рамки вокруг зоны приема, чтобы пользователь понимал, что именно она служит «ловушкой» перетаскиваемого элемента (рис. 10.1).

Рис. 10.1. Перетаскивание желтого квадрата на синий квадрат

Событие **dragover** непрерывно срабатывает, пока перетаскиваемый объект находится поверх элемента, играющего роль зоны приема, поэтому не стоит постоянно включать отображение красной рамки (это создало бы ненужную нагрузку на браузер). Следует запрещать поведение по умолчанию для этого элемента в зависимости от типа перетаскиваемого объекта. Вот почему мы вводим строки **preventDefault()** и **return false**.

Наконец, нам нужно подключить объект **dataTransfer** с данными, необходимыми браузеру для обработки действий по перетаскиванию. Поэтому мы меняем обработчик события **dragstart**:

```
foobar.addEventListener("dragstart", function(evt) {
    this.style.border = "3px dotted #000"; // черная пунктирная рамка

    evt.dataTransfer.effectAllowed = "move";
    evt.dataTransfer.setData("Text", this.id);
}, false);
```

Свойство **effectAllowed** управляет визуальным эффектом (чаще всего это вид указателя мыши), который браузер создает в ответ на тип происходящего события перетаскивания (перемещение, копирование и т. п.).

Метод **setData(...)** сообщает механизму перетаскивания в браузере, какие данные из перетаскиваемого объекта должен «поймать» целевой элемент, также называемый *зоной приема*. Здесь мы указываем, что передается только свойство **id** исходного элемента, которое в дальнейшем используется для его фактического перемещения.

Теперь нужно определить обработчик события **dragend**, который будет очищать визуальные эффекты, и указать событие **drop**, выполняющее работу по непосредственному перемещению элемента:

```
foobar.addEventListener("dragend", function(evt) {
    this.style.border = ""; // удаляем рамку
}, false);

catcher.addEventListener("drop", function(evt) {
    if (evt.preventDefault) evt.preventDefault();
    if (evt.stopPropagation) evt.stopPropagation();

    this.style.border = ""; // удаляем рамку зоны приема

    var id = evt.dataTransfer.getData("Text"); // получаем id
    var elem = document.getElementById(id);
    elem.parentNode.removeChild(elem); // удаляем элемент
    this.appendChild(elem); // добавляем элемент на зону приема

    return false;
}, false);
```

В обработчике события **drop** мы сначала получаем данные, которые были переданы в зону приема. В нашем случае это свойство **id** исходного перетаскиваемого элемента. После этого мы удаляем указанный элемент из его текущего местоположения и добавляем обратно, но в новое место внутри зоны приема. Результат показан на рис. 10.2.

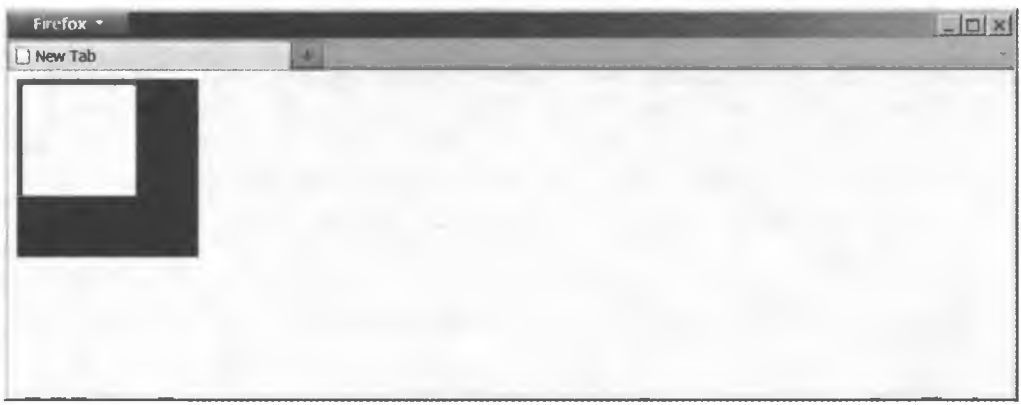

Рис. 10.2. Результат перетаскивания желтого квадрата на синий квадрат

Обсуждение

«Родной» API перетаскивания в HTML5 весьма функционален, но и очень сложен. Как вы могли убедиться, для реализации такого перетаскивания на своем сайте вам не потребуется присоединять комплексную библиотеку, однако и тривиальным решение нельзя назвать. Тем не менее сложность обеспечивает высокую гибкость.

Попробовав выполнить предыдущий код самостоятельно, вы сразу заметите (если это не очевидно из представленных рисунков), что, если удерживать нажатой кнопку мыши и перемещать по экрану указатель, желтый квадрат останется неподвижным.

В разных браузерах это может визуализироваться по-разному, однако чаще всего желтый квадрат во время перетаскивания останется на своем месте. Визуальное изменение включает только новый указатель мыши, подсказывающий пользователю, что события перетаскивания действительно выполняются.

Получается, что элемент, который мы собираемся перетаскивать, остается на прежнем месте, и это наверняка противоречит ожиданиям. Логичнее было бы увидеть действие, аналогичное перетаскиванию окна программы на Рабочем столе компьютера: окно «прилипает» к указателю мыши и перемещается вместе с ним, подчиняясь движениям руки.

С другой стороны, если вы попробуете перетащить значок по Рабочему столу в Windows, то он не будет следовать за указателем мыши, а переместится на новое место, когда вы отпустите кнопку мыши и прекратите перетаскивание. Это больше похоже на то, как осуществляется перетаскивание в HTML5.

Необходимо понимать, что «родная» функциональность перетаскивания предназначена для переноса объектов всевозможного рода, не каждому из которых соответствует визуальное представление, такое как желтый квадрат. Здесь мы выходим за рамки фактического перемещения элементов на странице. Суть перетаскивания заключается в том, что мы берем ссылку на какой-то объект и переносим ее на другой объект, соединяя их в событии перетаскивания. Вам предоставляется право решать, по какому признаку происходит соединение и как должны реагировать различные объекты.

Далее перечислены события, связанные с «родной» функциональностью перетаскивания.

- **dragstart** — вызывается в самом начале переноса «перетаскиваемого» элемента.
- **dragend** — вызывается в конце события перетаскивания — как успешного, так и отмененного.

- **dragenter** — вызывается на зоне приема, когда перетаскиваемый объект впервые оказывается над ней.
- **dragleave** — вызывается, когда перетаскиваемый объект покидает зону приема.
- **dragover** — непрерывно вызывается, когда перетаскиваемый объект находится над зоной приема.
- **drop** — вызывается, когда событие перетаскивания завершается отпусканием элемента над зоной приема.

Иногда перетаскивание означает простое копирование невидимых атрибутов из одного элемента в другой. В иных случаях оно может инициировать какое-то действие, например удаление. Так происходит, когда вы перетаскиваете элемент на значок Корзины. В нашем примере мы помещаем желтый квадрат внутрь синего, и, когда событие перетаскивания завершается, мы фактически изменяем местоположение желтого квадрата.

Поскольку желтый квадрат можно увидеть, то кажется вполне логичным, что он должен двигаться по экрану, пока мы осуществляем перетаскивание (как при перемещении объекта на Рабочем столе). Однако здесь типичный подход JavaScript не действует, потому что во время перетаскивания событие **mousemove** не срабатывает. Глобальное событие **drag** срабатывает на документе, но невозможность точно определять координаты в разных браузерах не позволяет применять его для позиционирования элемента.

Если вам необходимо фактически перемещать элемент по экрану в процессе перетаскивания, то лучше вообще не прибегать к «родной» функциональности, по крайней мере, пока. Надеюсь, с этим недостатком разработчикам спецификации вскоре удастся справиться. Однако до тех пор нам остается только искать помощи среди проверенных схем и/или библиотек, имитирующих перетаскивание на базе событий мыши.

Еще один расширенный вариант применения «родной» функциональности перетаскивания, который вскоре обещает появиться, — это возможность переносить элементы между окнами или между браузером и Рабочим столом, и наоборот. Например, можно перетянуть файл с Рабочего стола и отпустить его в зоне приема на веб-странице. Ответная реакция на подобное перетаскивание определяется точно так же, как во фрагментах кода выше. Единственное различие заключается в том, что у вас на странице нет предназначенного для перетаскивания элемента с атрибутом **draggable**. Например:

```
catcher.addEventListener("drop", function(evt) {
    if (evt.preventDefault) evt.preventDefault();
    if (evt.stopPropagation) evt.stopPropagation();

    this.style.border = ""; // удаляем рамку зоны приема
```

продолжение ↛

```
    var files_array = evt.dataTransfer.files;
    // Теперь у вас есть ссылка на файл (или несколько), который пользователь
    // перетащил на вашу страницу. Сделайте с ним что-нибудь интересное!

    return false;
}, false);
```

Как видите, «родная» функциональность перетаскивания фокусируется на переносе данных между двумя элементами, а не на перемещении какого-то элемента из одного места экрана в другое. И это открывает огромные возможности. В примере выше в свойстве **dataTransfer.files** мы получаем список ссылок на файлы, которые пользователь выделил и перетащил в приложение. Визуального оформления у такой операции по умолчанию нет — это лишь перетаскивание данных (ссылок на файлы) с Рабочего стола на элемент на нашей странице.

У нас есть ссылка на файл в системе пользователя. Но что с ней можно сделать? Об этом мы поговорим в рецепте 10.7.

Рассмотренная функциональность наверняка претерпит еще множество изменений, прежде чем будет принята и стандартизирована во всех браузерах. Однако очень приятно знать, что это случится совсем скоро!

Дополнительная информация

Подробнее о «родной» функциональности перетаскивания HTML5 рассказывается в учебнике по адресу http://www.html5rocks.com/en/tutorials/dnd/basics/.

10.4. Рабочие процессы

Проблема

Необходимо выполнить сложную и длительную задачу JavaScript, не блокируя пользовательский интерфейс в браузере.

Решение

Нужно запустить выполнение задачи JavaScript в отдельном потоке, для чего следует использовать API **Worker** — процесс, называемый Web Worker, или «рабочий процесс».

Этот API создает специальную среду для кода JavaScript, где тот выполняется в собственном потоке, отделенном от основного пользовательского интерфейса страницы. Получается, что интерфейс не замирает, даже если код JavaScript выполняется достаточно долго.

Чтобы проверить, поддерживаются ли в браузере рабочие процессы, можно использовать следующую строку кода:

```
var webworkers_support = !!window.Worker;
```

Теперь создадим простой демонстрационный код на основе рабочих процессов. Мы инициализируем большой двумерный массив со случайными числами — эта задача может выполняться довольно долго и приведет к заметной задержке в работе пользовательского интерфейса. Подобный массив случайных чисел можно применять для окрашивания случайных пикселов на элементе **canvas**, подробнее о котором рассказывается в главе 9. Нужную нам задачу выполняют два вложенных цикла **for**:

```
var data = [];
for (var i=0; i<1500; i++) {
    data[i] = [];
    for (var j=0; j<1500; j++) {
        data[i][j] = Math.random();
    }
}
```

Ничего особенного здесь не происходит. Однако инициализация такого массива, включающая 2,25 млн (1500×1500) операций, с большой вероятностью может заблокировать пользовательский интерфейс на период от 2 до 30 секунд, в зависимости от браузера и возможностей устройства.

Решение станет намного изящнее и поможет избежать блокировки пользовательского интерфейса, если мы вынесем операции в отдельный поток — рабочий процесс — и просто подождем уведомления об их завершении, прежде чем продолжать работу с данными массива.

Для этого поместим код из предыдущего фрагмента в отдельный файл (например, **init_array.js**) и заключим его в обработчик события **onmessage**:

```
self.onmessage = function(evt) {
    var data = [];
    for (var i=0; i<1500; i++) {
        data[i] = [];
        for (var j=0; j<1500; j++) {
            data[i][j] = Math.random();
        }
    }

    self.postMessage(data);
    data = null; // удаляем нашу копию данных для очистки памяти
};
```

Это и есть код, определяющий рабочий процесс. Сначала мы приказываем рабочему процессу слушать событие **message**, благодаря которому тот узнает, когда запускаться. Затем рабочий процесс выполняет длительное вычисление. В конце он отправляет сообщение (в нашем примере — массив данных) обратно главной странице, используя для этого **postMessage(...)**. Рабочий процесс также можно запустить из другого процесса, и обмен сообщениями между ними будет происходить точно так же.

В отличие от обычных операций JavaScript, в которых объекты передаются по ссылке, данные, передаваемые в сообщениях рабочих процессов, копируются. Это удваивает количество памяти, необходимой для передачи информации. Однако с большинством типов данных это не создает особых проблем.

С другой стороны, в примере мы применяем большой массив, поэтому нельзя не учитывать значительный объем используемой памяти, так как это может привести к сбоям, например, на мобильных устройствах. Для экономии памяти удаляйте переменные, содержащие большие данные, сразу после того, как необходимость в них отпадает.

На главной странице пользовательского интерфейса мы создаем рабочий процесс и указываем, к какому файлу он будет относиться. Затем мы настраиваем перехват события **message** и получение сообщения (инициализированного массива) от закончившего задачу рабочего процесса. Наконец, мы запускаем рабочий процесс, отправляя пустое сообщение в **postMessage()**:

```
var worker = new Worker("init_array.js");

worker.onmessage = function(evt) {
    alert("Инициализация массива данных завершена!");
    var data = evt.data;
};

worker.postMessage(); // приказываем рабочему процессу начать выполнение
```

Обсуждение

Рабочие процессы очень полезны для выгрузки сложных или длительных задач в отдельный поток, так как эта функциональность недоступна в чистом JavaScript. Если рабочие процессы в определенном браузере не поддерживаются, то код приходится выполнять в основном потоке JavaScript и придумывать, как справляться с неизбежными задержками. В некоторых ситуациях можно разбивать длинный код на более мелкие фрагменты и выполнять их по отдельности с небольшими паузами. За время паузы пользовательский интерфейс успеет обновиться. Например:

```
function doNextChunk() {
    var done_yet = false;
    for (var i=0; i<500; i++) { // выполнить 500 за раз
        // делаем что-нибудь
        // когда работа сделана, установить done_yet = true
    }
    if (!done_yet) setTimeout(doNextChunk,0);
    else alert("Наконец-то, все сделано!");
}

doNextChunk();
```

Используя шаблон **setTimeout(..., 0)**, мы выполняем 500 итераций длительного цикла, делаем небольшую паузу (достаточную для того, чтобы пользовательский интерфейс успел обновиться), затем продолжаем работу, проходя очередные 500 итераций, и т. д. Такой подход обеспечивает более высокую производительность, чем при единовременном выполнении всего кода, когда пользовательский интерфейс может замереть на неопределенное время. Однако это все же менее эффективно, чем решение на основе рабочих процессов.

Создавая рабочий процесс, вы определяете связь между основным кодом JavaScript на странице и изолированным фрагментом кода, выполняющимся в отдельном потоке. Две стороны этой связи общаются между собой асинхронно, отправляя и получая сообщения. Для этого применяются перехват события **message** и метод **postMessage(...)**.

Асинхронный вызов Ajax, отправленный на сервер с помощью XMLHttpRequest (XHR), — это приблизительно то же самое, что отправка асинхронных сообщений рабочему процессу и получение от него ответов.

Коммуникационный интерфейс Web Worker также поддерживает отправку и получение сообщений об ошибках. Для того чтобы сообщить об ошибке внутри рабочего процесса, используйте вызов JavaScript **throw**:

```
self.onmessage = function(evt) {
    var data = [];
    for (var i=0; i<1500; i++) {
        data[i] = [];
        for (var j=0; j<1500; j++) {
            data[i][j] = Math.random();
            if (data[i][j] == 0) {
                throw "В моем массиве не должно быть нулей!";
            }
        }
    }

    self.postMessage(data);
    data = null; // удаляем нашу копию данных для освобождения памяти
};
```

Для получения сообщения об ошибке от рабочего процесса настройте перехват события **error**:

```
var worker = new Worker("init_array.js");

worker.onerror = function(err) {
    alert("Во время инициализации массива произошла ошибка.");
    throw err; // необязательно
};

worker.onmessage = function(evt) {
    alert("Инициализация массива данных завершена!");
    var data = evt.data;
};

worker.postMessage();
```

Рабочий процесс выполняется отдельно от главной страницы, и единственное, что ему доступно, — это обмен сообщениями с ней. Получается, что он не может обращаться к DOM для чтения и модификации какой бы то ни было информации. Помимо этого, процессу не под силу выполнять задачи, рассчитанные на пользовательский интерфейс, такие как вызов диалогового окна **alert(...)**.

Однако у рабочего процесса есть несколько полезных преимуществ. Например, он может обращаться к объекту **navigator** для идентификации пользовательского агента (браузера), в котором выполняется. Он также может загружать в себя сценарии с помощью команды **importScripts(...)**:

```
if (navigator.userAgent.test(/MSIE/)) { // "вынюхивание" пользовательского
                                        // агента – плохая практика!!
    importScripts("ie_helper.js");
}
self.onmessage = function(evt) {
    /* ... */
};
```

Функция loadScripts(...) параллельно загружает один или несколько сценариев, однако всегда исполняет их в запрошенном порядке. Кроме того, она выполняется синхронно, то есть блокирует остальную деятельность рабочего процесса до тех пор, пока сама не завершит загрузку и исполнение сценариев.

Как вы только что видели, один рабочий процесс может создать другой рабочий процесс. В том же коде, где процесс создается, можно и завершать его, вызывая **terminate()** для конкретного экземпляра рабочего процесса.

Наконец, рабочие процессы могут использовать задержки и интервалы, задействуя методы **setTimeout(...)**, **clearTimeout(...)**, **setInterval(...)**

и **clearInterval(...)**. Эти методы удобно применять, например, для создания рабочих процессов, которые периодически выполняются в фоновом режиме, уведомляя страницу о каждом своем запуске:

```
self.onmessage = function(evt) {
    setInterval(function(){
        self.postMessage(Math.random()); // отправляем обратно случайное число
    }, 60*60*1000); // выполнять один раз в час
};
```

Дополнительная информация

Спецификацию рабочих процессов W3C вы найдете на сайте http://dev.w3.org/html5/workers/.

10.5. Веб-сокеты

Проблема

Вы хотели бы создать постоянный двусторонний канал коммуникации между своим приложением и сервером, чтобы и браузер, и сервер могли по необходимости отправлять друг другу данные и получать их.

Решение

Сегодня большинство браузеров по умолчанию поддерживают возможность установки двунаправленного сокетного соединения с сервером через API **WebSocket**. Это означает, что обе стороны (браузер и сервер) способны отправлять и получать данные. Очень часто веб-сокеты используются при создании интерактивных игр в режиме реального времени, для передачи котировок ценных бумаг и другой биржевой информации, в клиентах чата и т. п.

Чтобы проверить, поддерживает ли браузер веб-сокеты, можно использовать следующий код:

```
var websockets_support = !!window.WebSocket;
```

Теперь создадим простое приложение с функциональностью чата, в котором пользователь сможет читать уже переданные сообщения и добавлять собственные.

На странице мы создадим поле для ввода текста, где пользователь будет печатать сообщение перед отправкой. У нас также будет область со списком

сообщений, отправленных в чат ранее. Нам не нужны такие возможности, как вход в систему и аутентификация; это будет простейший чат с возможностью отправки и получения сообщений:

```
<!DOCTYPE html>
<html>
<head>
<title>Наш чат</title>
<script src="chatroom.js"></script>
</head>
<body>
<h1>Наш чат</h1>

<div id="chatlog"></div>

<input id="newmsg" /><br />
<input type="button" value="Отправить сообщение" id="sendmsg" />
</body>
</html>
```

Рассмотрим код JavaScript в файле **chatroom.js**:

```
var chatcomm = new WebSocket("ws://something.com/server/chat");

chatcomm.onmessage = function(msg) {
    msg = JSON.parse(msg); // декодируем JSON в объект

    var chatlog = document.getElementById("chatlog");
    var docfrag = document.createDocumentFragment();
    var msgdiv;

    for (var i=0; i<msg.messages.length; i++) {
        msgdiv = document.createElement("div");
        msgdiv.appendChild(document.createTextNode(msg.messages[i]));
        docfrag.appendChild(msgdiv);
    }

    chatlog.appendChild(docfrag);
};

chatcomm.onclose = function() {
    alert("Соединение с чатом потеряно. Для повторного подключения обновите
страницу.");
};

document.getElementById("sendmsg").addEventListener("click", function(){
    var newmsg = document.getElementById("newmsg");

    chatcomm.send(newmsg.value); // отправка сообщения на сервер
    newmsg.value = ""; // очистка поля для ввода текста
}, false);
```

Предлагаю рассмотреть этот код по частям. В самом начале мы создаем сокет и передаем ему адрес местоположения на нашем сервере. Серверный URL-адрес в примере использует протокол ws://, а не более распространенный

http://, с которым все хорошо знакомы. Это означает, что мы будем работать со специальным протоколом, на базе которого веб-сокеты обмениваются сообщениями между клиентом и сервером.

После этого мы настраиваем в объекте сокета два приемника событий: **onmessage** и **onclose**. Назначение обработчика **onclose** очевидно — он срабатывает при закрытии соединения.

Мы не будем рассматривать серверную реализацию нашего демонстрационного приложения, так как это не относится к теме книги. Однако вы можете найти множество учебников и программных проектов, с помощью которых вам будет совсем не сложно написать серверную часть чата на любом подходящем языке, включая PHP, JavaScript (node.js), Java и т. д.

На серверной стороне чата требуется реализовать только простейшие действия по отправке и получению данных. Нужный JavaScript-код будет очень похож на тот, который мы используем для реализации клиента. Для приложения базового уровня даже нет необходимости сохранять сообщения на сервере: он обязан публиковать все сообщения, которые получает через сокеты, чтобы их видели клиенты, подключенные к нему в данный момент.

Обработчик **onmessage** получает строку данных (в примере мы ожидаем, что это будет JSON) и передает ее по частям в объект **message**, содержащий массив из одного или нескольких сообщений (состоящих из простого текста). Обработчик в цикле проходит по полученным сообщениям и добавляет их в журнал чата в порядке получения.

Наконец, в коде определяется обработчик события **click** для кнопки **Send Message** (Отправить сообщение). Когда нажатие произошло, обработчик считывает то, что было введено в текстовом поле, и отправляет эти данные на сервер через метод **send(...)**.

Обсуждение

Разумеется, такой тип функциональности появился давно. Со времени изобретения Ajax и объекта **XMLHttpRequest** (XHR) разработчики обмениваются данными между браузером и сервером. Кроме того, применяются другие подходы, включая создание невидимого объекта Flash и использование возможностей сокетной коммуникации Flash.

Однако что касается варианта с XHR, довольно неэффективно создавать новое соединение для каждого фрагмента данных, которые требуется отправить из браузера на сервер. Точно так же нежелательно создавать занимающий много памяти объект Flash, чтобы воспользоваться его возможностями

сокетной коммуникации. Теперь вполне понятно, почему веб-сокеты были с такой радостью приняты в компанию «HTML5 и его друзей».

В технологии веб-сокетов отправка и получение сообщений выглядит как нечто среднее между XHR и рабочими процессами (Web Worker), о которых мы говорили в предыдущем рецепте.

Для того чтобы приложение на базе веб-сокетов работало, обе стороны — браузер и сервер — должны говорить на одном языке, то есть применять стандартизованный и согласованный протокол (аналогично тому, как для обычных веб-страниц используется протокол HTTP). Однако за последние пару лет в ходе разработки этот протокол подвергся множеству экспериментов и изменений.

Несмотря на то что ситуация начинает стабилизироваться, спецификация веб-сокетов еще находится в довольно неустойчивом состоянии, поэтому вы должны удостовериться, что ваш сервер поддерживает самую новую версию протокола, иначе браузер не сможет общаться с ним без ошибок.

У экземпляра объекта **WebSocket** аналогично XHR есть свойство **readyState**, позволяющее проверять состояние подключения. Оно принимает одно из следующих постоянных значений:

- **{worker}.CONNECTING** (числовое значение **0**) — соединение еще не установлено;
- **{worker}.OPEN** (числовое значение **1**) — соединение открыто, возможен обмен данными;
- **{worker}.CLOSING** (числовое значение **2**) — соединение закрывается;
- **{worker}.CLOSED** (числовое значение **3**) — соединение закрыто (или его не удалось открыть).

Экземпляр объекта **WebSocket** может запускать следующие события:

- **open** — вызывается после того, как открывается соединение;
- **message** — вызывается, когда от сервера было получено сообщение;
- **error** — вызывается, когда в сокете происходит ошибка (при отправке или получении данных;
- **close** — вызывается, когда соединение закрывается.

Для каждого из этих событий можно с помощью **addEventListener(...)** добавить получателя или же установить соответствующий обработчик — **onopen**, **onmessage**, **onerror** или **onclose** — непосредственно на экземпляре объекта рабочего процесса.

Если веб-сокеты не поддерживаются, то в приложении необходимо предусмотреть альтернативное решение, обеспечивающее ту же функциональность,

либо, по крайней мере, вежливо уведомить пользователя, что нужные функции в его браузере недоступны. К счастью, это очень просто сделать.

Поскольку единообразия в поддержке веб-сокетов браузерами пока не наблюдается, лучше всего для реализации этой функциональности использовать библиотеку, такую как Socket.io (http://socket.io). Она пытается обеспечить работу через API веб-сокетов или, если он недоступен, прибегает к различным альтернативным способам обмена данными.

Необходимо также знать, на каком уровне поддерживается функциональность веб-сокетов с точки зрения серверных ресурсов. Традиционные веб-запросы захватывают только выделенные ресурсы сервера и лишь на долю секунды, благодаря чему сервер в состоянии обслуживать огромный объем веб-трафика без пересечений и не сталкиваясь с проблемой нехватки ресурсов.

С другой стороны, сокеты требуют к себе повышенного внимания со стороны сервера, поэтому при высокой нагрузке возможны проблемы с доступностью ресурсов. Настройка и архитектура вашего сервера в значительной степени определяют, насколько успешно вы сможете применять возможности веб-сокетов и должны соответствовать требованиям приложения.

Дополнительная информация

Подробнее о библиотеке Socket.io рассказывается на домашней странице проекта: http://socket.io.

10.6. История

Проблема

Вы хотели бы управлять в своем веб-приложении очередью адресов, связанных с кнопками Вперед/Назад, а также списком URL в адресной строке браузера.

Решение

В HTML5 к объекту **window.history**, представляющему известный API **History**, добавлено несколько важных усовершенствований.

Чтобы проверить, поддерживает ли браузер улучшенный API **History**, используйте следующую строку:

```
var history_support = !!(window.history && window.history.pushState);
```

Обычно при изменении URL в адресной строке браузер инициирует новый запрос к серверу, чтобы получить указанную страницу. Однако современные сложные веб-приложения чаще всего загружают свежую информацию с помощью Ajax, не обновляя страницу полностью. В результате веб-приложения стараются не менять URL в адресной строке только потому, что не хотят, чтобы браузер обновлял страницу.

Для того чтобы изменить URL в адресной строке, но при этом избежать перезагрузки страницы, используйте метод **history.pushState(...)**. Он обновляет URL-адрес и создает специальную завязанную на состоянии запись в истории браузера. Это означает, что если пользователь нажмет в своем браузере кнопку Назад, то вместо загрузки предыдущей страницы браузер запустит новое событие **popstate**, а ваше приложение может отреагировать на него переводом страницы в предыдущее состояние.

Новый URL-адрес, который вы передаете методу pushState() или replaceState(), должен относиться к тому же источнику (домену и т. д.), что и текущая страница. В противном случае API вернет ошибку. Вы можете поменять в URL путь, имя файла, строку запроса и хэш-значение, но не протокол/схему, домен и порт.

Бессмысленно и небезопасно смешивать URL из разных источников в очереди состояний. Для перехода между разными источниками используйте обычные приемы управления адресами и историей.

Рассмотрим пример, в котором эти две новые возможности совместно помогают управлять навигацией вперед и назад, ориентируясь только на состояния страницы (и без необходимости полностью обновлять ее контент). Вы также узнаете, как сделать так, чтобы в адресной строке браузера всегда отображался актуальный URL.

Код из примера отслеживает состояние элемента: видим он или нет. Это состояние отражается в навигационном стеке браузера и в адресной строке, благодаря чему актуальный URL можно скопировать и вставить в любой документ или добавить в список закладок:

```
<html>
<head>
<title>Пример истории</title>
<script>
function showText(updateHistory) {
    document.getElementById("long_desc").style.display = "block";
    if (updateHistory) history.pushState(null, null, "?show");
}
function hideText(updateHistory) {
    document.getElementById("long_desc").style.display = "none";
```

```
    if (updateHistory) history.pushState(null, null,
                            location.href.replace(/\?show/, ""));
}
function toggleText() {
    var elem = document.getElementById("long_desc");
    if (elem.style && elem.style.display == "none") showText(true);
    else hideText(true);
}
function manageText() {
    if (location.href.match(/\?show/)) showText();
    else hideText();
}

window.addEventListener("popstate", manageText, false);
window.addEventListener("DOMContentLoaded", function(){
    document.getElementById("toggle").addEventListener("click", function(e){
        toggleText();
        e.preventDefault();
        return false;
    }, false);

    manageText();
}, false);
</script>
</head>

<body>
<p>Это короткое описание.</p>
<a id="toggle" href="#">переключить</a>
<p id="long_desc">Это длинное описание, которое можно показать или спрятать.</p>
</body>
</html>
```

Если вы выполните этот код и пощелкаете на ссылке **переключить**, то увидите, что отображение абзаца с длинным описанием действительно включается и выключается. Вы также заметите, что когда абзац видим, в URL-адресе присутствует параметр **?show**, а когда спрятан — этого параметра нет. Кроме того, навигационные кнопки позволяют циклически переходить между этими состояниями, показывая и пряча абзац в зависимости от того, на какую страницу вы перешли.

Попробуйте скопировать URL-адрес с параметром **?show** и вставить его на новую вкладку браузера. Вы увидите, что на открывшейся странице абзац действительно отображается — состояние остается в URL-адресе, как мы и хотели. Код из этого примера сохраняет изменения состояния страницы в очереди, связанной с навигационными кнопками **Вперед**/**Назад** браузера. В одних приложениях это желаемое поведение, но в других недопустимо заполнение истории множеством промежуточных изменений состояния.

В таком случае вместо **pushState(...)** можно использовать **replaceState(...)**. Этот метод заменяет текущую запись состояния в истории навигации новой

записью, указывающей желаемое состояние. Если сделать это в предыдущем примере, то код будет выглядеть так:

```
// ...

function showText(updateHistory) {
    document.getElementById("long_desc").style.display = "block";
    if (updateHistory) history.replaceState(null, null, "?show");
}
function hideText(updateHistory) {
    document.getElementById("long_desc").style.display = "none";
    if (updateHistory) history.replaceState(null, null,
                                location.href.replace(/\?show/, ""));
}

// ...
```

Выполнив обновленный код, вы увидите, что переключение щелчком на ссылке и URL-адрес работают так же, как и раньше. Единственное отличие заключается в отсутствии истории состояний, то есть теперь невозможно переключаться между состояниями с помощью навигационных кнопок.

Обсуждение

Браузеры уже давно поддерживают API **History**. Отличие HTML5 от предыдущих спецификаций состоит в появлении новой усовершенствованной функциональности методов **pushState(...)**, **replaceState(...)** и **popstate**.

До того как в браузеры был добавлен улучшенный API **History** из спецификации HTML5, разработчикам приходилось реализовывать описанные выше возможности с помощью хэш-значения в URL (адрес с таким значением заканчивается на **#здесь|что-то|написано**).

Большинство браузеров обрабатывают хэш-значение одинаково: если вы поменяете его для текущей страницы, то обозреватель сохранит это состояние в истории навигации, обновит URL в адресной строке, но не станет запрашивать новую страницу с сервера. На первый взгляд, это именно то, что нужно. Однако существует несколько хитростей, усложняющих получение единообразных и надежных результатов при работе с изменениями хэш-значений.

В частности, не все старые браузеры поддерживают событие **hashchange**, весьма полезное для мониторинга состояния URL-хэша в том случае, когда пользователь копирует и вставляет в адресную строку URL с хэш-значением. В отсутствие поддержки такого события разработчику приходится опрашивать URL-хэш с использованием таймера. К счастью, с этой неразберихой обычно помогают справиться различные библиотеки. Одна из наиболее полезных называется History.js (https://github.com/balupton/history.js). Она использует усовершенствованный API **History** из HTML5, а при неудаче автоматически возвращается к управлению URL-хэшем.

В предыдущем коде мы сохраняем в URL-адресе простое состояние **?show**. Это удобно для копирования и вставки (и добавления в закладки), так как состояние целиком описывается контентом URL-адреса и поэтому легко восстанавливается.

Если же вам приходится управлять сложным набором состояний, а копирование, вставка и добавление в закладки не очень важны, то с каждой записью истории можно сохранять более богатый набор информации. Все сложное состояние целиком сохраняется в записи истории, а затем, когда пользователь возвращается к нему нажатием кнопки Назад браузера, оно восстанавливается и отправляется приложению через обработчик события **popstate**.

Первый параметр методов **pushState(...)** и **replaceState(...)** — это объект **state** произвольного уровня сложности. Единственное требование — он должен быть сериализован в строковое значение. Например:

```
window.addEventListener("popstate", function(e){
    alert("Данные текущего состояния: " + JSON.stringify(e.state));
}, false);

window.pushState({foo:"bar"}, null, "?foobar");
window.pushState({bar:"baz"}, null, "?barbaz");
history.back(); // вызываем popstate для возврата
                // к странице/состоянию "?foobar"
```

В настоящее время браузеры не поддерживают второй параметр, представляющий собой «заголовок» нового состояния, поэтому вместо него нужно передавать значение null или пустую строку.

Дополнительная информация

Подробнее об использовании API **History** рассказывается в следующей статье на сайте MDC: https://developer.mozilla.org/en/DOM/Manipulating_the_browser_history. Более развернутую информацию о History.js вы найдете в архиве github: https://github.com/balupton/history.js.

10.7. Локальные файлы

Проблема

Необходимо сделать так, чтобы пользователь мог открыть файл изображения из локальной файловой системы и сделать с ним что-то на веб-странице, например просмотреть или загрузить на сервер.

Решение

До появления HTML5 единственный вариант взаимодействия пользователя с локальной файловой системой обеспечивался элементом **<input type="file">**. Для кода JavaScript, использующегося на странице, эта функциональность оказывалась абсолютно непрозрачной, так как невозможно было что-либо узнать о выбранном файле или как-то осмысленно его обработать.

В HTML5 теперь есть API **FileReader**, позволяющий извлечь ссылку на локальный файл и прочитать его контент прямо на веб-страницу. Для проверки, поддерживает ли браузер усовершенствованный API **FileReader**, используйте следующую строку кода:

```
var history_support = typeof FileReader != "undefined";
```

В рецепте 10.3 вы узнали, как получить ссылку на один или несколько локальных файлов с применением «родной» функциональности перетаскивания. Аналогичным образом извлекаются и ссылки на локальные файлы, выбранные пользователем в элементе **<input type="file">**:

```
<p>Выберите файл изображения:</p>
<input type="file" id="file_selector" />

<script>
var file_selector = document.getElementById("file_selector");
file_selector.addEventListener("change", function(){
    var files_array = this.files;
    // Теперь у вас есть ссылка на выбранные пользователем файлы.
    // Сделайте с ними что-нибудь примечательное!
}, false);
</script>
```

Каким бы образом вы ни извлекали ссылку на локальный файл, например, с изображением, получив ее, вы можете прочитать содержимое файла с помощью экземпляра **FileReader**:

```
function read_image_file(file) {
    var reader = new FileReader();
    reader.onload = function(e){
        var image_contents = e.target.result;
        // теперь у вас есть содержимое файла
    };
    reader.readAsDataURL(file);
}
```

В примере мы получаем содержимое файла в виде *URI-данных* (файл в кодировке base64). Теперь его можно вывести в элементе **img**. Соберем весь код вместе:

```
<p>Выберите файл изображения:</p>
<input type="file" id="file_selector" />
```

```
<script>
var file_selector = document.getElementById("file_selector");
file_selector.addEventListener("change", function(){
    var files_array = this.files;
    // мы разрешили выбирать только один файл
    if (files_array[0].type.match(/image/)) { // это файл изображения
        read_image_file(files_array[0]);
    }
}, false);

function read_image_file(file) {
    var reader = new FileReader();
    reader.onload = function(e){
        var image_contents = e.target.result;
        var img = document.createElement("img");
        img.src = image_contents;
        document.body.appendChild(img);
    };
    reader.readAsDataURL(file);
}
</script>
```

В этом фрагменте кода предполагается, что пользователь выбрал только один файл. Однако элемент <input type="file"> теперь поддерживает атрибут multiple, позволяющий выбирать одновременно несколько файлов. По этой причине мы в результате получаем массив ссылок на файлы, а не единственную ссылку на один файл.

Если вы хотите позволить пользователю загрузить выбранный файл изображения на сервер, просто отправьте содержимое файла на удаленную машину в вызове XHR Ajax:

```
<p>Выберите файл изображения:</p>
<input type="file" id="file_selector" />
<input type="button" id="upload" value="Upload Image" disabled />

<script>
var file_selector = document.getElementById("file_selector");
file_selector.addEventListener("change", function(){
    var files_array = this.files;
    // мы разрешили выбирать только один файл
    if (files_array[0].type.match(/image/)) { // это файл изображения
        read_image_file(files_array[0]);

        file_selector.disabled = true; // теперь элемент управления,
                                       // предназначенный для выбора файла,
                                       // нужно отключить
        var upload = document.getElementById("upload");
        upload.disabled = false;
        upload.addEventListener("click", function(){
            upload_file(files_array[0]);
```

продолжение ⇩

```
        }, false);
    }
}, false);

function upload_file(file) {
    var xhr = new XMLHttpRequest();
    xhr.setRequestHeader("Content-Type", "multipart/form-data");
    xhr.setRequestHeader("X-File-Name", file.fileName);
    xhr.setRequestHeader("X-File-Size", file.fileSize);
    xhr.setRequestHeader("X-File-Type", file.type);
    xhr.open("GET", "image_upload.php");
    xhr.send(file);
}

function read_image_file(file) {
    var reader = new FileReader();
    reader.onload = function(e){
        var image_contents = e.target.result;
        var img = document.createElement("img");
        img.src = image_contents;
        document.body.appendChild(img);
    };
    reader.readAsDataURL(file);
}
</script>
```

Обратите внимание, что теперь появилась возможность считывать имя файла, его размер и тип, и эти данные можно отправить на сервер вместе с содержимым файла. Ничего другого, относящегося к HTML5, мы здесь не делаем — файл загружается стандартными методами XHR.

Обсуждение

HTML5 предоставляет API **FileReader**, благодаря которому мы можем считывать содержимое локальных файлов в системе пользователя и применять его на своих веб-страницах.

В примере выше показано, как вывести на страницу эскиз изображения из файла пользователя и отправить (то есть загрузить) контент на сервер с помощью Ajax. Разумеется, с данными из файла можно выполнять и массу других действий. Например, в API **FileReader** есть метод **readAsBinaryString(...)**, благодаря которому можно получить содержимое файла в виде двоичной строки. Если вам известен формат считываемого файла, то вы легко выполните над этими данными различные операции.

Еще один пример: если поместить данные изображения в элемент **img** (как показано выше), то, применяя рекомендации из рецепта 9.5, можно вывести

это изображение на элементе **canvas**, а затем применить к нему разнообразные цветовые и геометрические преобразования (см. рецепты 9.6 и 9.7).

В настоящее время доступ к локальным файлам ограничен только операцией чтения, причем она должна быть инициирована пользователем: например, он сам должен перетащить файл на веб-страницу или выбрать один или несколько файлов с помощью элемента **<input type="file">**. Вероятно, с точки зрения безопасности это наилучший вариант, так как разрешать веб-страницам записывать данные в локальную файловую систему слишком рискованно, хотя это было бы здорово!

Кроме того, помните, что возможность доступа к локальным файлам из браузеров пока находится на стадии разработки, поэтому необходимо тщательно тестировать ее в предпочтительных браузерах вашей целевой аудитории и обеспечивать обходные пути на тот случай, если она окажется недоступной или будет работать не так, как ожидалось.

Дополнительная информация

Подробнее о взаимодействии с локальными файлами через веб-страницу рассказывается в следующей статье на сайте MDC: https://developer.mozilla.org/en/Using_files_from_web_applications.

Приложение. Ресурсы HTML5

Сравнение поддержки HTML5 в разных браузерах (http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML_5)).

Не все браузеры одинаковы. На этой странице вы можете проверить, насколько хорошо в различных браузерах поддерживаются возможности HTML5.

Сравнение библиотек JavaScript (http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks).

В этой статье «Википедии» сравниваются возможности разных библиотек JavaScript, упрощающих построение веб-приложений. Помимо прочего, из всеобъемлющего списка на этой странице вы узнаете, насколько хорошо каждая библиотека справляется с базовыми функциями обеспечения доступности веб-страниц и с поддержкой ARIA.

«Погружение в HTML5» («Dive into HTML5» — http://diveintohtml5.info/).

Марк Пилгрим, чьи статьи стали общественным достоянием, а код используется в книге «HTML5 с нуля» (http://shop.oreilly.com/product/9780596806033.do), раскрывает мельчайшие подробности спецификации HTML5 и особенности поддержки этого стандарта разными браузерами.

«Библиотека стандартных решений HTML5» («HTML5 Boilerplate» — http://html5boilerplate.com/).

Это название несколько неточно, поскольку речь идет не только о HTML5. Однако небольшая неточность названия — единственный недостаток проекта. «HTML5 Boilerplate» — это дипломный проект, посвященный настройке и поддержке эффективных шаблонов веб-страниц.

Шпаргалка по идентификаторам и названиям классов HTML5 (http://oli.jp/2008/html5-class-cheatsheet/).

Создавать шаблоны удобнее всего на примере предыдущих работ. Обратите внимание на это исследование, посвященное наиболее используемым значениям атрибутов **id** и **class**.

Реализация HTML5 в браузерах (http://wiki.whatwg.org/wiki/Implementations_in_Web_browsers).

На этой вики-странице вы найдете список возможностей HTML5 с указанием того, в каких браузерах они поддерживаются.

Демонстрационные приложения HTML5 (http://html5demos.com/).

Коллекция демонстрационных приложений, собранная разработчиком JavaScript Реми Шарпом.

«Доктор HTML5» (http://html5doctor.com/).

Создатели сайта HTML5 Doctor начали заниматься разъяснением спецификации и тестированием различных ее частей еще до появления IE9 и Chrome. Этот сайт постоянно обновляется, и здесь вы всегда найдете самую свежую информацию и обсуждение проблем, с которыми сталкиваются веб-дизайнеры при работе с HTML5.

«Перезагрузка HTML5» (http://html5reset.org/).

Базовый набор файлов HTML и CSS для кодирования проектов с использованием HTML5. Вы найдете ресурсы для самых разных браузеров, включая старые версии Internet Explorer.

«Крутой HTML5» (http://www.html5rocks.com/).

HTML5 Rocks — это игровая площадка, где собраны разнообразные примеры и учебники, создаваемые в Google.

HTML: справочник языка разметки (http://dev.w3.org/html5/markup/spec.html).

Это домашняя страница спецификации HTML5, созданной W3.

Modernizr (http://www.modernizr.com/).

Распознавание браузера, через который посетитель зашел на ваш сайт, — это подход из 1990-х годов. В новом тысячелетии разработчики веб-приложений проверяют, поддерживает ли браузер нужную функциональность, не интересуясь его названием. Эта библиотека JavaScript помогает проверять наличие таких возможностей HTML5, как геолокация, вебсокеты и даже свойства CSS3.

«Когда можно использовать» («When Can I Use» — http://caniuse.com/).

Если вам необходимо проверить, поддерживает ли браузер нужную возможность HTML5, проще всего воспользоваться справочником «When Can I Use».

Об авторах

Кристофер Шмитт работает с Сетью с 1993 года. Он возглавляет компанию по производству мультимедийной продукции Heatvision.com, Inc. и живет в Остине, штат Техас, США. Кристофер часто выступает с лекциями, посвященными веб-дизайну и стандартизованной разработке, на различных конференциях, включая OSCON, SXSW Interactive, AIGA In Control и CSS Summit. Он написал такие книги, как «Адаптация к веб-стандартам: CSS и Ajax для больших сайтов» (*Adapting to Web Standards: CSS and Ajax for Big Sites*, издательство New Riders), «Профессиональное использование CSS: каскадные таблицы стилей для веб-дизайна» (*Professional CSS: Cascading Style Sheets for Web Design*, издательство Wrox) и «Сборник рецептов CSS» (*CSS Cookbook*, издательство O'Reilly, http://shop.oreilly.com/product/9780596005764.do). Его личный сайт находится по адресу http://christopherschmitt.com.

Кайл Симпсон — разработчик пользовательских интерфейсов. Живет в Остине, штат Техас, США. Один из его главных интересов — проектирование взаимодействия с пользователем и, в частности, оптимизация пользовательского интерфейса. Кайл стремится сделать его максимально удобным, эффективным, безопасным и масштабируемым. Он считает JavaScript самым совершенным языком программирования и непрерывно ищет новые задачи, которые можно решить с его помощью. Если чего-то невозможно добиться с использованием JavaScript или веб-технологий, то Кайла это не интересует. Он поддерживает несколько проектов с открытым кодом, включая LABjs, HandlebarJS/BikechainJS и flXHR, а также вносит огромный вклад в развитие SWFObject.