БИБЛИОТЕКА ПРОГРАММИСТА

Брайан Хоган

HTML5 и CSS3

Веб-разработка по стандартам нового поколения

Pragmatic Bookshelf

ПИТЕР®

HTML5 and CSS3

Develop with Tomorrow's Standards Today

Brian P. Hogan

The Pragmatic Bookshelf
Raleigh, North Carolina Dallas, Texas

БИБЛИОТЕКА ПРОГРАММИСТА

Брайан Хоган

HTML5 и CSS3

Веб-разработка по стандартам нового поколения

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

Брайан Хоган

HTML5 и CSS3. Веб-разработка по стандартам нового поколения

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Юрченко*
Ведущий редактор	*Ю. Сергиенко*
Художественный редактор	*Л. Адуевская*
Корректор	*В. Листова*
Верстка	*Е. Егорова*

ББК 32.988.02-018
УДК 004.738.5

Хоган Б.
Х68 HTML5 и CSS3. Веб-разработка по стандартам нового поколения. — СПб.: Питер, 2012. — 272 с.: ил.

ISBN 978-5-459-00592-9

HTML5 и CSS3 — будущее веб-разработки, но не обязательно ждать будущего, чтобы начать применять эти стандарты уже сегодня. Хотя спецификации этих языков еще находятся в разработке, большинство современных браузеров и мобильных устройств поддерживают HTML5 и CSS3. Эта книга поможет вам использовать HTML5 и CSS3 прямо сейчас, применяя все богатые возможности, появившиеся в новых веб-стандартах.

Вы научитесь применять новую разметку HTML5, разрабатывать улучшенные интерфейсы для форм ввода данных, узнаете, как добавлять аудио, видео и векторную графику на веб-страницы без использования Flash. Вы увидите, как хранение данных на стороне клиента в автономном режиме кэширования может кардинально улучшить скорость загрузки веб-страниц и как в этом помогают простые решения, доступные в CSS3. Каждый раздел книги сопровождается многочисленными примерами, а для каждой описанной функции читателю предстоит создать небольшой учебный пример.

ISBN 978-1934356685 англ.
ISBN 978-5-459-00592-9

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 14.10.11. Формат 70x100/16. Усл. п. л. 21,930. Тираж 2000. Заказ 26587.
Отпечатано по технологии CtP в ОАО «Первая Образцовая типография», обособленное подразделение «Печатный двор». 197110, Санкт-Петербург, Чкаловский пр., 15.

Оглавление

ЧАСТЬ IV
ПРИЛОЖЕНИЯ

Предисловие

Три месяца в веб-программировании — все равно что год реального времени.

У веб-разработчиков время идет в ускоренном темпе: ведь мы постоянно слышим о чем-то новом. Еще год назад HTML5 и CSS3 казались делом будущего, но уже сегодня компании используют эти технологии в своей работе, потому что браузеры (Google Chrome, Safari, Firefox и Opera) начали реализовывать отдельные части этой спецификации.

Технологии HTML5 и CSS3 закладывают основу для следующего поколения веб-приложений. Сайты, созданные с применением этих технологий, более просты в разработке и сопровождении и более удобны для пользователей. В HTML5 появились новые элементы для определения структуры сайта и встроенного контента, которые избавляют нас от необходимости использовать дополнительную разметку или плагины. CSS3 предоставляет расширенные селекторы, графические усовершенствования и улучшенную поддержку работы со шрифтами, которые позволят нам сделать наши сайты более привлекательными без применения графической замены шрифтов, сложного кода JavaScript и графических инструментов. Приложения Ajax станут более доступными для людей с физическими недостатками, а поддержка автономной работы позволит строить приложения, не нуждающиеся в подключении к Интернету.

В этой книге вы узнаете о том, как использовать HTML5 и CSS3 прямо сейчас, несмотря на то что браузеры ваших пользователей еще не поддерживают всех возможностей. Но сначала мы немного поговорим о HTML5 и некоторых модных словечках.

HTML5: платформа и спецификация

HTML5 — спецификация, описывающая некоторые новые теги и разметку, а также ряд замечательных Javascript API, но она попала в вихрь рекламной шумихи и обещаний. К сожалению, стандарт HTML5 развился до платформы HTML5, что создало невероятную путаницу среди разработчиков, клиентов и даже авторов книг. В некоторых случаях фрагменты спецификации CSS3 (например, тени, градиентные заливки и трансформации) называются «HTML». Создатели браузеров стараются превзойти друг друга по тому, в какой степени их продукты поддерживают «HTML5». Заказчики начинают задавать странные вопросы: «Мой сайт будет написан на HTML5, правда?»

В основной части этой книги мы сосредоточимся на спецификациях HTML5 и CSS3 и на использовании описанных в них методов. В завершающей части книги рассматривается группа сопутствующих спецификаций, которые когда-то были частью HTML5, а сейчас используются на разных платформах. К их числу относятся спецификации Web SQL Databases, Geolocation и Web Sockets. Хотя *формально* эти технологии не относятся к HTML5, в сочетании с HTML5 и CSS3 они позволяют добиться потрясающего эффекта.

Как это делается

Каждая глава книги посвящена конкретной группе задач, которые могут решаться с использованием HTML5 и CSS3. В каждой главе приводится сводка и таблица с перечислением тегов, возможностей и концепций, рассмотренных в этой главе. Основной материал каждой главы состоит из разделов, в которых излагается некоторая концепция, после чего рассматривается процесс построения простого примера с использованием этой концепции. Главы книги группируются по темам. Вместо деления книги на части по технологиям (HTML5 и CSS3), нам показалось более разумным сгруппировать материал в соответствии с решаемыми задачами.

В каждом разделе присутствует подраздел «Обходное решение», в котором рассказано, что делать, если браузер пользователя еще не поддерживает HTML5 и CSS3. Работоспособность обходных решений обеспечивается разными средствами, от сторонних библиотек до

наших собственных плагинов jQuery. Разделы можно читать в любом порядке.

Каждая глава завершается разделом «Перспективы», в нем обсуждается возможность применения концепции по мере ее более широкого распространения.

Основное внимание в книге уделяется тому, что можно использовать уже сегодня. Многие возможности HTML5 и CSS3 еще не получили широкого распространения. Они более подробно описаны в главе 11.

О чем рассказано в книге

Книга открывается кратким обзором HTML5 и CSS3. Мы рассмотрим несколько новых структурных тегов, используемых для семантического описания содержимого страницы. Затем мы поработаем с формами; вам представится возможность воспользоваться новыми полями и возможностями форм — такими, как автофокус и заполняющий текст. Далее мы немного поэкспериментируем с новыми селекторами CSS3, и вы узнаете, как применять стили к элементам без включения в контент дополнительной разметки.

Затем мы займемся поддержкой аудио и видео и использованием холста (canvas) для рисования геометрических фигур. Кроме того, в этой части рассматривается работа с тенями, градиентами и трансформациями CSS3, а также операции со шрифтами.

В последней части представлены клиентские средства HTML5 — Web Storage, Web SQL Databases и режим автономной работы для построения приложений, работающих на стороне клиента. Мы используем Web Sockets для построения простого чата, и вы увидите, как в HTML5 обеспечивается передача отправки сообщений и данных между доменами. Также вам представится возможность поэкспериментировать с Geolocation API и историей просмотра. Книга завершается описанием некоторых возможностей, которые пока не приносят особой пользы, но будут играть важную роль в недалеком будущем.

В приложении А перечислены всех возможности HTML5, представленные в книге, с указанием глав, в которых эти возможности описаны более подробно. В книге широко используется технология jQuery, поэтому в приложении Б приводится краткий вводный курс. Также в небольшом приложении В объясняются особенности кодирования аудио- и видеофайлов для использования с HTML5.

Исходные требования

Книга написана прежде всего для веб-разработчиков с хорошим знанием HTML и CSS. Она пригодится и тем, кто еще делает первые шаги в освоении этих технологий, но новичкам я бы порекомендовал сначала прочитать *Designing with Web Standards* [Zel09] и мою книгу *Web Design for Developers* [Hog09]. Также предполагается, что читатель хотя бы в общих чертах разбирается в JavaScript и jQuery[1] (последняя технология будет использована для реализации многих обходных решений. Все основные методы jQuery, используемые в книге, рассматриваются в приложении Б.

Для тестирования кода, приведенного в книге, потребуется браузер Firefox 3.6, Google Chrome 5, Opera 10.6 или Safari 5. А еще лучше иметь все эти браузеры, потому что каждый из них несколько отличается в нюансах от других.

Также вам понадобится возможность тестирования сайтов в Internet Explorer для проверки работоспособности обходных решений. Если ваши примеры должны тестироваться в разных версиях Internet Explorer, используйте IETester for Windows — этот пакет поддерживает IE 6, 7 и 8 в одном приложении. Если вы не работаете в Windows, подумайте об использовании виртуальной машины (например, VirtualBox или VMWare) или стороннего сервиса вроде CrossBrowserTesting[2] или MogoTest[3].

Сетевые ресурсы

На сайте книги[4] размещены ссылки на форумы, списки опечаток и ссылка на исходный код всех примеров. Читатели электронной версии могут щелкнуть на сером квадратике над фрагментом кода, чтобы загрузить его напрямую.

Если вы обнаружили ошибку, пожалуйста, сообщите о ней на соответствующей странице, чтобы мы могли решить проблему. Если вы используете электронную версию книги, в нижнем колонтитуле страницы имеются ссылки для удобной отправки сообщений об ошибках.

Наконец, обязательно посетите блог этой книги[5]. В нем будут публиковаться сопутствующие материалы, обновления и рабочие примеры.

Готовы? Отлично! Наше знакомство с HTML5 и CSS3 начинается.

---

[1] http://www.jquery.com/
[2] http://crossbrowsertesting.com/
[3] http://www.mogotest.com/
[4] http://www.pragprog.com/titles/bhh5/
[5] http://www.beyondhtml5andcss3.com/

От издательства

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

Мы будем рады узнать ваше мнение!

Все исходные тексты, приведенные в книге, вы можете найти на вебсайте издательства «Питер»: http://www.piter.com.

1 Обзор HTML5 и CSS3

HTML5[1] и CSS3[2] — не просто два новых стандарта, предложенных комитетом W3C (World Wide Web Consortium) и его рабочими группами. Это следующее поколение ежедневно используемых технологий, созданное для того, чтобы вам было проще и удобнее строить современные веб-приложения. Прежде чем погружаться в подробности HTML5 и CSS3, давайте немного поговорим о преимуществах HTML5 и CSS3, а также о некоторых проблемах, с которыми мы столкнемся при использовании этих технологий.

1.1. Платформа веб-разработки

Многие новые возможности HTML направлены на совершенствование платформы для построения веб-приложений. HTML5 предоставляет в распоряжение разработчика много новых инструментов для улучшения пользовательского интерфейса, от более содержательных тегов и улучшенных средств межсайтовых и межоконных коммуникаций до анимации и улучшенной мультимедийной поддержки.

Более содержательная разметка

В каждой версии HTML появляется новая разметка, но еще никогда не было столько дополнений, напрямую связанных с описанием контента.

---

[1] Спецификация HTML5 находится по адресу http://www.w3.org/TR/html5/

[2] Спецификация CSS3 разбита на несколько модулей. Информация о его текущем состоянии доступна по адресу http://www.w3.org/Style/CSS/current-work

Элементы для определения заголовков, завершителей, навигационных областей, боковых панелей и статей рассматриваются в главе 2. Также вы узнаете о датчиках, индикаторах выполнения и возможностях разметки данных с применением пользовательских атрибутов.

Мультимедийные возможности без зависимости от плагинов

Теперь для использования видео, аудио и векторной графики вам уже не понадобится Flash или Silverlight. Видеопроигрыватели на базе Flash относительно просты в использовании, но они не работают на мобильных устройствах Apple. Чтобы не терять весьма значительную долю рынка, необходимо научиться использовать альтернативные видеотехнологии, не требующие Flash. В главе 7 показано, как использовать аудио- и видеосредства HTML5 с эффективными обходными решениями.

Расширенные возможности создания приложений

Для создания более мощных, более интерактивных веб-приложений разработчики перепробовали множество разных технологий, от элементов ActiveX до Flash. В HTML5 реализован целый ряд замечательных возможностей, которые во многих случаях могут полностью обойтись без сторонних технологий.

Передача информации между доменами

Браузеры не позволяют сценариям одного домена влиять на сценарии другого домена или взаимодействовать с ними. Это ограничение защищает пользователей от междоменных сценарных атак, которые порой создают всевозможные неприятности ничего не подозревающим посетителям сайтов.

Однако такое ограничение запрещает выполнение любых сценариев, даже если мы написали их сами и уверены в том, что им можно доверять. В HTML5 имеется обходное решение этой проблемы — безопасное и одновременно просто реализуемое. Вы увидите, как оно работает, в рецепте 24 «Передача информации между доменами» на с. 213.

Web Sockets

В HTML5 включена поддержка технологии Web Sockets, реализующей долгосрочное подключение к серверу. Вместо постоянного опроса исполнительной подсистемы для получения информации о ходе выполнения

ваша веб-страница подключается к сокету, а исполнительная подсистема доставляет оповещения пользователям. Мы немного поэкспериментируем с этой возможностью в рецепте 25, «Чат на базе Web Sockets» (с. 220).

Хранение данных на стороне клиента

Обычно HTML5 рассматривается как веб-технология, но с появлением прикладных интерфейсов (API) Web Storage и Web SQL Database появилась возможность создания браузерных приложений, которые хранят все данные на машине клиента. Пример использования этих API приведен в главе 9, «Работа с данными на стороне клиента».

Улучшения интерфейса

Пользовательский интерфейс является важной частью веб-приложений. Нам приходится ежедневно идти на всяческие ухищрения, чтобы заставить браузеры работать так, как мы хотим. Чтобы определить стилевое оформление таблицы или закруглить углы, приходится использовать либо библиотеки JavaScript, либо добавлять массу дополнительной разметки для применения стилей. Благодаря HTML5 и CSS3 эта практика становится делом прошлого.

Улучшения форм

HTML5 предоставляет в распоряжение разработчика усовершенствованные элементы пользовательского интерфейса. В течение многих лет мы использовали JavaScript и CSS для построения ползунков, календарей для выбора даты и палитр для выбора цвета. В HTML5 они стали обычными элементами — такими же, как раскрывающиеся списки, флажки и переключатели. Об использовании новых элементов рассказано в главе 3, «Новые возможности веб-форм». И хотя поддержка новых элементов реализована еще не во всех браузерах, о ней не стоит забывать, особенно если вы разрабатываете веб-приложения. Кроме удобства использования, не требующего библиотек JavaScript, существует и другое преимущество — улучшенная доступность для пользователей с физическими недостатками. Экранные дикторы и другие специализированные браузеры могут реализовать эти элементы особым образом, чтобы с ними было удобно работать людям с ограниченными возможностями.

Улучшения доступности

Использование новых элементов HTML5 упрощает работу с контентом в таких специализированных программах, как экранные дикторы. Например, область навигации по сайту намного проще найти, если вы ищете тег `nav` вместо конкретного тега `div` или неупорядоченного списка. Завершители, боковые панели и другие структурные элементы легко перемещаются или вовсе исключаются из просмотра. Упрощение разбора страниц в целом также упростит работу со страницей для людей, использующих вспомогательные технологии. Кроме того, возможность определения ролей в новых атрибутах элементов поможет экранным дикторам работать с ними. В главе 5, «Улучшение доступности», вы научитесь использовать эти новые атрибуты, чтобы современные экранные дикторы могли работать с ними.

Расширенные селекторы

Селекторы CSS3 позволяют определить нечетные и четные строки в таблице, все установленные флажки и даже последний абзац в группе. Более сложные задачи решаются с меньшим объемом кода и разметки. В главе 4 показано, как эффективно использовать эти селекторы.

Визуальные эффекты

Тени, отбрасываемые текстом и изображениями, придают веб-странице визуальную глубину, а градиенты создают иллюзию объема. CSS3 позволяет добавлять тени и градиенты без использования фоновой графики и дополнительной разметки. Трансформации используются для закругления углов, деформации и поворота элементов. Все эти возможности рассматриваются в главе 8.

1.2. Обратная совместимость

Одна из самых веских причин для немедленного перехода на HTML заключается в том, что разметка работает в большинстве существующих браузеров. Вы можете начать использовать HTML5 прямо сейчас, даже в Internet Explorer 6, и постепенно перерабатывать свою разметку. Она даже будет проходить валидацию W3C (условно, конечно, потому что стандарты продолжают развиваться).

Каждый, кто когда-либо работал с HTML или XML, уже сталкивался с объявлением `doctype`. Оно сообщает программам валидации и редакторам, какие теги и атрибуты будут использоваться в документе и как должен быть сформирован документ. Объявление `doctype` также используется многими браузерами для определения того, как браузер должен воспроизводить страницу. Действительное объявление `doctype` обычно заставляет браузер воспроизводить страницу в «режиме соответствия стандартам».

По сравнению с довольно пространным объявлением XHTML 1.0 Transitional, используемым на многих сайтах:

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

объявление `doctype` в HTML5 выглядит до смешного просто:

html5_why/index.html

```
<!DOCTYPE html>
```

Поместите его в начало документа — и с этого момента вы используете HTML5.

Конечно, вы не сможете использовать многие новые элементы HTML5, которые еще не поддерживаются вашим браузером, но документ будет проверяться на валидность как разметка HTML5.

1.3. Тернистый путь в будущее

Повсеместному переходу на HTML5 и CSS3 мешают различные препятствия. Некоторые из них очевидны, с другими дело обстоит сложнее.

КЕКС И ГЛАЗУРЬ

Я люблю кексы. Пожалуй, пирожные все же лучше, но и кексы очень хороши. Лично я предпочитаю кексы с глазурью.

Разрабатывая веб-приложения, следует помнить, что красивые пользовательские интерфейсы и изощренный код JavaScript — не более чем глазурь. Ваш сайт должен быть хорош и без них. Как и в случае с кексом, вам понадобится «основание» для нанесения глазури.

Я встречал людей, которые не любят глазурь. Они соскребают ее с кекса. Я также встречал людей, которые по тем или иным причинам используют веб-приложения, отключив JavaScript.

Приготовьте этим людям хороший кекс. А затем нанесите на него глазурь.

ВОПРОС/ОТВЕТ

Но мне нравятся самозакрывающиеся теги XHTML. Смогу ли я их использовать?

Конечно, сможете! Многие разработчики любят XHTML из-за более жестких требований к разметке. Документы XHTML заставили разработчиков заключать атрибуты в кавычки, использовать самозакрывающиеся контентные теги, записывать имена атрибутов в нижнем регистре, а также способствовали введению правильно сформированной разметки в World Wide Web. Переход на HTML5 не означает, что вам придется отказываться от старых привычек. Документы HTML5 будут действительными при использовании как синтаксиса HTML5, так и синтаксиса XHTML, но вы должны понимать последствия от использования самозакрывающихся тегов.

Как правило, веб-серверы выдают страницы HTML с типом MIME text/html, потому что Internet Explorer не умеет правильно обрабатывать тип application/xml+xhtml MIME, ассоциируемый со страницами XHTML. По этой причине браузеры обычно отсекают самозакрывающиеся теги, так как они не считались действительной разметкой HTML до выхода HTML5. Предположим, вы используете самозакрывающийся тег script над div:

```
<script language="javascript" src="application.js" />
<h2>Help</h2>
```

Браузер удалит косую черту самозакрывающегося тега, а подсистема вывода решит, что h2 находится в незакрытом теге script! Именно поэтому теги script кодируются явно закрываемым тегом, хотя самозакрывающийся тег и считается действительной разметкой XHTML.

Помните о возможности возникновения подобных проблем при использовании самозакрывающихся тегов в документах HTML5, потому что они будут выдаваться с типом MIME text/html. За дополнительной информацией об этой и других проблемах обращайтесь по адресу http://www.webdevout.net/articles/beware-of-xhtml#myths.

Internet Explorer

Internet Explorer в настоящее время имеет самую большую пользовательскую базу, а версии 8 и ниже имеют очень слабую поддержку HTML5 и CSS3. IE9 улучшает ситуацию, но эта версия еще не получила широкого распространения. Это не означает, что мы не должны использовать HTML5 и CSS3 на своих сайтах. Вы можете заставить свои сайты работать в Internet Explorer, но они будут работать не так, как версии, разработанные для Chrome и Firefox. Просто вам придется реализовать обходные решения, чтобы не злить пользователей и не терять клиентов.

Доступность

Наши сайты должны быть доступны для любых пользователей, даже если они испытывают сложности при восприятии видео- и аудиоинформации, используют старые браузеры или медленные подключения или просматривают сайт с мобильных устройств. В HTML5 появился ряд новых элементов — таких как `audio`, `video` или `canvas`. У аудио- и видеоконтента с доступностью бывали определенные затруднения, но элемент `canvas` создает новые проблемы. Он предназначен для создания векторных изображений в документах HTML на JavaScript. Это создает проблемы не только для лиц с ограниченными зрительными возможностями, но и для 5 % пользователей Интернета, отключающих JavaScript в своих браузерах[1].

В нашем стремлении к новым технологиям не забывайте о доступности. Предоставьте подходящие обходные решения для этих новых элементов HTML5 — по аналогии с тем, как бы вы сделали для пользователей Internet Explorer.

Устаревшие теги

В HTML5 появилось много новых элементов, но спецификация также объявляет устаревшими некоторые стандартные элементы, которые могут присутствовать в ваших веб-страницах[2]. Уберите их, прежде чем двигаться вперед.

Прежде всего исчезли некоторые элементы представления. Если они встречаются в вашем коде, избавьтесь от них! Замените их семантически правильными элементами и придайте им нужный вид средствами CSS:

- `basefont`;
- `big`;
- `center`;
- `font`;
- `s`;
- `strike`;
- `tt`;
- `u`.

---

[1] http://visualrevenue.com/blog/2007/08/eu-and-us-javascript-disabled-index.html
[2] http://www.w3.org/TR/html5-diff/

Некоторые из этих тегов встречаются относительно редко, но теги `font` и `center` присутствуют во многих страницах, для работы с которыми используются визуальные редакторы вроде Dreamweaver.

Кроме элементов представления, была исключена поддержка фреймов. Фреймы всегда пользовались популярностью в корпоративных веб-приложениях: PeopleSoft, Microsoft Outlook Web Access и даже специализированных порталах. Несмотря на широкое использование, фреймы создавали столько проблем с доступностью и удобством использования, что от них было решено отказаться. Таким образом, исчезли следующие элементы:

- `frame`;
- `frameset`;
- `noframes`.

Продумайте возможность структурирования своих интерфейсов без фреймов, стандартными средствами CSS или с использованием JavaScript. Если вы используете фреймы для того, чтобы одни и те же заголовки, завершители и области навигации присутствовали на каждой странице приложения, вы сможете добиться того же эффекта при помощи инструментов, предоставленных вашими средствами веб-разработки. Еще некоторые элементы исчезают из-за появления более совершенных заменителей:

- `acronym` заменяется на `abbr`;
- `applet` заменяется на `object`;
- `dir` заменяется на `ul`.

Кроме устаревших элементов, недействительными стали многие атрибуты. К их числу относятся следующие атрибуты представления:

- `align`;
- атрибуты `link`, `vlink`, `alink` и `text` тега `body`;
- `bgcolor`;
- `height` и `width`;
- атрибут `scrolling` элемента `iframe`;
- `valign`;
- `hspace` и `vspace`;
- атрибуты `cellpadding`, `cellspacing` и `border` тега `table`.

браузеры тоже изменятся; это может привести к появлению устаревших, неработоспособных сайтов.

Во время написания книги тени `box-shadow` были исключены из CSS3 и заново добавлены в спецификацию, а модификация протокола Web Sockets привела к полному нарушению клиентско-серверных коммуникаций.

Если следить за развитием HTML4 и CSS3 и оставаться в курсе происходящего, все будет хорошо. Большинство возможностей, рассматриваемых в книге, будет работать без изменений в течение долгого времени.

Если вы столкнулись с чем-то, что не работает в одном из ваших целевых браузеров, заполните пробел «на ходу» при помощи JavaScript и Flash. У вас получится надежное решение, подходящее для всех пользователей, а с течением времени JavaScript и другие обходные решения можно будет удалить без изменения реализации.

Впрочем, не стоит заглядывать слишком далеко в будущее — лучше перейдем к непосредственной работе с HTML5. В стандарте появился целый набор новых структурных тегов, с которыми вы познакомитесь в следующей главе.

I

Усовершенствования пользовательского интерфейса

2 Новые структурные теги и атрибуты

В начальных главах этой книги мы поговорим о том, как использовать возможности HTML5 и CSS для совершенствования интерфейсов, с которыми работают пользователи. Вы увидите, как создать улучшенные формы, как легко определять стилевое оформление таблиц и улучшить доступность страниц для использования с вспомогательными устройствами. Мы рассмотрим генерирование контента для удобной работы со стилевыми таблицами печати и возможности редактирования «на месте» с использованием атрибута `contenteditable`. А для начала посмотрим, как улучшить структуру наших страниц при помощи новых элементов HTML5.

Существует одно серьезное «заболевание», которому подвержены многие веб-разработчики нашего времени. Вокруг нас бушует эпидемия *дивита* — хронического синдрома, который заставляет людей упаковывать элементы в лишние теги `div` с идентификаторами `banner`, `sidebar`, `article`, `footer` и т. д. Заболевание в высшей степени заразное. Разработчики быстро подхватывают дивит друг от друга, а так как лишние теги `div` не видны невооруженным глазом, легкие случаи дивита могут оставаться незамеченными годами.

Типичное проявление дивита выглядит так:

```
<div id="navbar_wrapper">
  <div id="navbar">
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/">Home</a></li>
    </ul>
  </div>
</div>
```

Здесь неупорядоченный список, уже являющийся блочным элементом[1], упакован в два тега `div`, которые также являются блочными элементами. Атрибуты `id` этих элементов-«оберток» сообщают нам, что они делают, но по крайней мере одну «обертку» можно убрать — результат от этого не изменится. Злоупотребление разметкой приводит к разрастанию страниц, усложнению их стилевого оформления и сопровождения.

Впрочем, не все потеряно. Спецификация HTML5 предоставляет лекарство от этой болезни — новые семантические теги, описывающие содержащийся в них контент. Так как многие разработчики включают в свои страницы боковые панели, заголовки/завершители и секции, в спецификацию HTML5 были включены новые теги, предназначенные для деления страницы на логические области. Давайте применим эти новые элементы на практике. Взяв на вооружение HTML5, мы сможем избавиться от симптомов дивита на всю оставшуюся жизнь.

Кроме новых структурных тегов мы также рассмотрим элемент `meter` и возможности использования пользовательских атрибутов HTML5, позволяющих встраивать данные прямо в элементы.

В этой главе рассматриваются следующие новые элементы и возможности[2]:

`<header>`

Определение заголовка страницы или раздела. [C5, F3.6, IE8, S4, O10]

`<footer>`

Определение завершителя страницы или раздела. [C5, F3.6, IE8, S4, O10]

`<nav>`

Определение области навигации страницы или раздела. [C5, F3.6, IE8, S4, O10]

`<section>`

Определение логической области страницы или группировка контента. [C5, F3.6, IE8, S4, O10]

---

[1] Напомним, что блочные элементы всегда начинаются с новой строки, тогда как встроенные (inline) элементы не требуют разрыва строки.

[2] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

`<article>`

Определение статьи (логически завершенного блока контента). [C5, F3.6, IE8, S4, O10]

`<aside>`

Определение вторичного или связанного контента. [C5, F3.6, IE8, S4, O10]

Пользовательские атрибуты данных

Возможность включения пользовательских данных в любой элемент с использованием схемы `data-`. [Все браузеры поддерживают чтение таких данных методом JavaScript `getAttribute()`]

`<meter>`

Представление величины, находящейся в заданном диапазоне. [C5, F3.5, S4, O10]

`<progress>`

Отображение информации о ходе некоторой операции в реальном времени. [Не поддерживался на момент издания книги.]

Рецепт 1. Реструктуризация блога с использованием семантической разметки

Контент, для которого актуальна структурная разметка, очень часто встречается в блогах. В страницах блогов используются заголовки и завершители, множественные схемы навигации (архивы, списки ссылок на другие блоги (блогроллы), внутренние ссылки) и, конечно, статьи или сообщения. Давайте воспользуемся разметкой HTML для построения макета главной страницы вымышленной компании AwesomeCo.

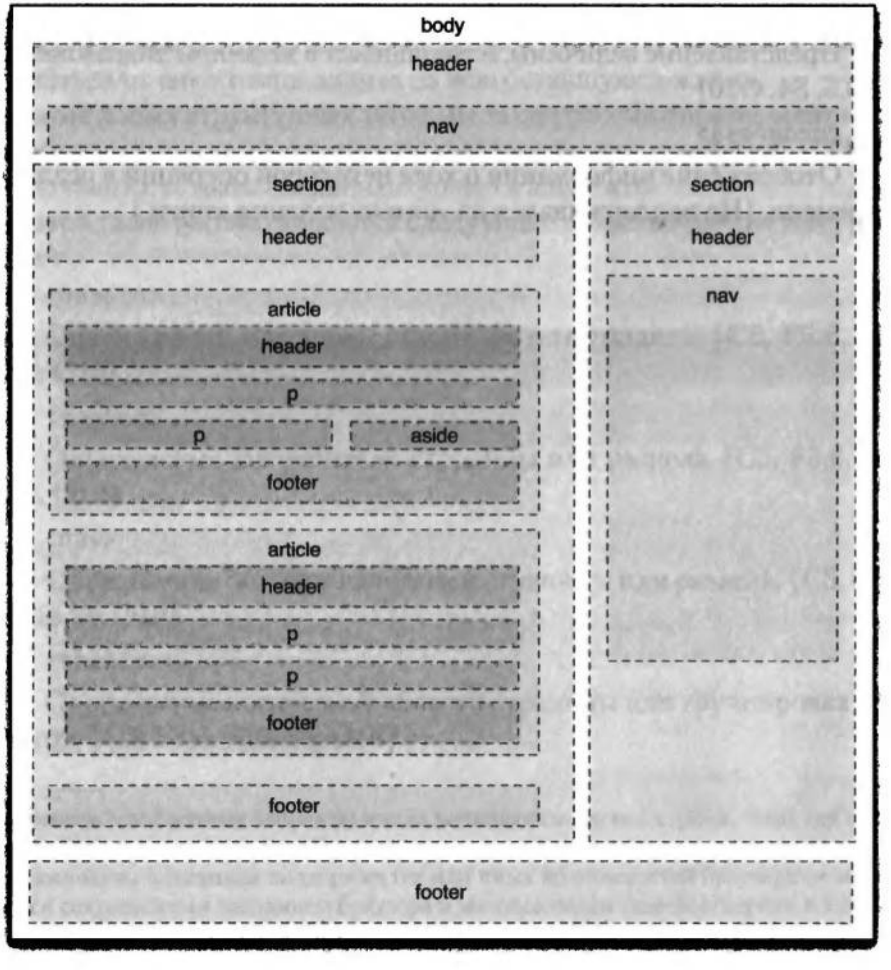


Рис. 2.1. Структура блога в семантической разметке HTML5

На рис. 2.1 изображена общая схема страницы. Она имеет вполне типичную для блогов структуру: основная заголовочная область с главным заголовком и расположенной под ним горизонтальной областью навигации. В главном разделе каждая статья также обладает заголовком и завершителем. Статьи также могут содержать выносные цитаты (pull quotes). Справа находится боковая панель с дополнительными элементами навигации. В области завершителя располагаются контактные данные и информация об авторских правах. В такой структуре нет ничего принципиально нового, не считая того, что на этот раз для описания этих областей вместо многочисленных тегов `div` будут использованы специальные теги HTML5.

Примерный вид готовой страницы показан на рис. 2.2.

Все начинается с правильной директивы doctype

Чтобы использовать новые элементы HTML5, необходимо сообщить информацию о новых тегах браузерам и валидаторам. Создайте новую страницу с именем *index.html* и включите в нее следующий шаблон HTML5:

html5newtags/index.html

```
1  <!DOCTYPE html>
2  <html lang="en-US">
3    <head>
4      <meta http-equiv="Content-Type" content="text/html;
                        charset=utf-8" />
5      <title>AwesomeCo Blog</title>
6    </head>
7
8    <body>
9    </body>
10 </html>
```

Взгляните на директиву `doctype` в строке 1 — это все, что необходимо для объявления типа документа HTML5. Если у вас уже имеется опыт создания веб-страниц, вероятно, вы привыкли к длинным и трудным для запоминания директивам `doctype` стандарта XHTML.

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

А теперь посмотрите на аналогичную директиву HTML5:

```
<!DOCTYPE html>
```

Она выглядит намного проще и легче запоминается.

Директива doctype выполняет сразу две функции. Во-первых, она помогает валидаторам определить, какие правила следует применять для проверки валидности кода. Во-вторых, она заставляет Internet Explorer версий 6–8 перейти в «режим соответствия стандартам» — это очень важно, если создаваемая вами страница должна работать во всех браузерах. Директива doctype для HTML5 выполняет обе функции, причем ее распознает даже Internet Explorer 6.

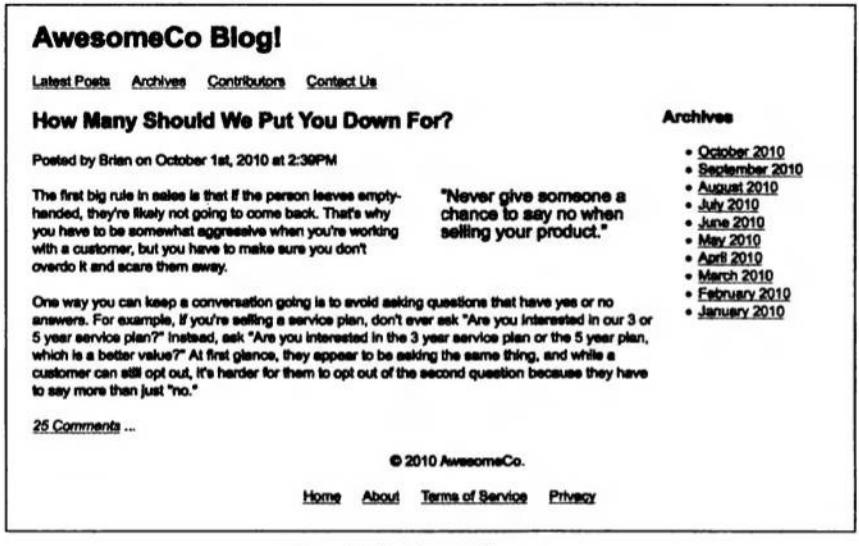


Рис. 2.2. Готовый макет

Заголовки

Заголовки (точнее, области заголовков — не путайте с заголовками `h1`, `h2` и т. д.) могут содержать любой контент, от логотипа фирмы до поля поиска. В заголовке нашего блога пока будет содержаться только название блога.

html5newtags/index.html

```
1 <header id="page_header">
2   <h1>AwesomeCo Blog!</h1>
3 </header>
```

Ничто не заставляет вас ограничиться всего одним заголовком на странице. Каждый конкретный раздел или статья может иметь свой заголовок, поэтому для однозначного определения элементов может быть удобно воспользоваться атрибутом `id` (как сделано в строке 1). Наличие уникального идентификатора упрощает стилевое оформление с применением CSS или поиск элементов в коде JavaScript.

СЕМАНТИЧЕСКАЯ РАЗМЕТКА

Семантическая разметка предназначена для описания контента. Если вы занимались созданием веб-страниц, вероятно, вы делили свои страницы на различные области (заголовок, завершитель, боковая панель и т. д.), чтобы вам было проще идентифицировать блоки страничного контента при применении таблиц стилей и других видов форматирования.

Семантическая разметка упрощает понимание смысла и контекста информации, размещенной на странице, — как для компьютеров, так и для людей. Новые теги разметки HTML5 — такие, как section, header и nav — помогут вам в решении этой задачи.

Завершители

Элемент `footer` определяет завершающую область документа или раздела. Завершители встречаются на многих сайтах, обычно в них содержится дата установления авторского права и информация о владельце сайта. В спецификации говорится, что документ может содержать несколько завершителей; таким образом, статьи нашего блога тоже могут иметь собственные завершители.

Давайте определим простой завершитель для нашей страницы. Так как страница может иметь несколько завершителей, мы присвоим завершителю идентификатор — по аналогии с тем, как это было сделано с заголовком. Это поможет однозначно идентифицировать этот конкретный завершитель при определении стилевого оформления для элемента и его потомков.

`html5newtags/index.html`

```
<footer id="page_footer">
  <p>&copy; 2010 AwesomeCo.</p>
</footer>
```

В данном случае завершитель содержит только дату установления авторского права. Однако завершители, как и заголовки, часто содержат и другие элементы, в том числе и навигационные.

html5newtags/index.html

```
<article class="post">
  <header>
    <h2>How Many Should We Put You Down For?</h2>
    <p>Posted by Brian on
      <time datetime="2010-10-01T14:39">October 1st, 2010 at
      2:39PM</time>
    </p>
  </header>
  <p>
    The first big rule in sales is that if the person leaves
    empty-handed, they're likely not going to come back. That's
    why you have to be somewhat aggressive when you're working
    with a customer, but you have to make sure you don't overdo
    it and scare them away.
  </p>
  <p>
    One way you can keep a conversation going is to avoid asking
    questions that have yes or no answers. For example, if
    you're selling a service plan, don't ever ask &amp;quot;Are you
    interested in our 3 or 5 year service plan?&amp;quot; Instead,
    ask &amp;quot;Are you interested in the 3 year service plan or
    the 5 year plan, which is a better value?&amp;quot;
    At first glance, they appear to be asking the same thing, and
    while a customer can still opt out, it's harder for them to
    opt out of the second question because they have to say more
    than just &amp;quot;no.&amp;quot;
  </p>
  <footer>
    <p><a href="comments"><i>25 Comments</i></a> ...</p>
  </footer>
</article>
```

В статьях могут использоваться элементы **header** и **footer**, что значительно упрощает описание этих конкретных разделов. Статья может быть разбита на несколько разделов при помощи элемента **section**.

ВОПРОС/ОТВЕТ

Чем статьи отличаются от разделов?

Раздел определяет логическую часть документа, а статья — его фактическое содержание (заметка, сообщение в блоге, новость и т. д.).

Новые теги описывают содержащийся в них контент. Раздел может содержать несколько статей, а статья может состоять из нескольких разделов.

Раздел можно сравнить со спортивной рубрикой в газете: рубрика может состоять из нескольких статей, а каждая статья может, в свою очередь, делиться на разделы. Некоторые разделы (такие, как заголовки и завершители) помечаются соответствующими тегами. Раздел — более общий элемент, который может использоваться для логической группировки других элементов.

Семантическая разметка предназначена для *смыслового* описания контента.

Дополнения и боковые панели

Иногда на странице размещается информация, дополняющая основной контент, — выносные цитаты, диаграммы, дополнительные мысли или сопутствующие ссылки. Для пометки таких элементов может использоваться новый тег `aside`.

html5newtags/index.html

```
<aside>
  <p>
    &quot;Never give someone a chance to say no when
    selling your product.&quot;
  </p>
</aside>
```

Выносная цитата размещается в элементе `aside`. Мы вложим дополнение в статью, чтобы оно находилось поблизости от своего контента.

html5newtags/index.html

```
<section id="posts">
  <article class="post">
    <header>
      <h2>How Many Should We Put You Down For?</h2>
      <p>Posted by Brian on
        <time datetime="2010-10-01T14:39">October 1st, 2010 at
        2:39PM</time>
      </p>
    </header>

    <aside>
      <p>
        &quot;Never give someone a chance to say no when
        selling your product.&quot;
```

```
      </p>
    </aside>
    <p>
      The first big rule in sales is that if the person leaves
      empty-handed, they're likely not going to come back.
      That's why you have to be somewhat aggressive when you're
      working with a customer, but you have to make sure you
      don't overdo it and scare them away.
    </p>
    <p>
      One way you can keep a conversation going is to avoid
      asking questions that have yes or no answers. For example,
      if you're selling a service plan, don't ever ask &amp;quot;Are
      you interested in our 3 or 5 year service plan?&amp;quot;
      Instead, ask &amp;quot;Are you interested in the 3 year
      service plan or the 5 year plan, which is a better
      value?&amp;quot;
      At first glance, they appear to be asking the same thing,
      and while a customer can still opt out, it's harder for
      them to opt out of the second question because they have
      to say more than just &amp;quot;no.&amp;quot;
    </p>
    <footer>
      <p><a href="comments"><i>25 Comments</i></a> ...</p>
    </footer>
  </article>
</section>
```

Осталось добавить раздел боковой панели.

Не путайте дополнения с боковыми панелями страниц!

В правой части нашего блога располагается панель со ссылками на архивы блога. Если вы думаете, что для определения боковой панели можно воспользоваться тегом `aside`, вы ошибаетесь. Такое решение возможно, но оно противоречит духу спецификации. Тег **`aside`** предназначен для пометки контента, относящегося к статье. Скажем, он хорошо подойдет для размещения сопутствующих ссылок, глоссария или выносных цитат.

Разметка боковой панели (список архивов за предыдущие периоды) будет состоять из тега `section` и еще одного тега `nav`.

html5newtags/index.html

```
<section id="sidebar">
  <nav>
    <h3>Archives</h3>
    <ul>
      <li><a href="2010/10">October 2010</a></li>
      <li><a href="2010/09">September 2010</a></li>
      <li><a href="2010/08">August 2010</a></li>
      <li><a href="2010/07">July 2010</a></li>
      <li><a href="2010/06">June 2010</a></li>
      <li><a href="2010/05">May 2010</a></li>
      <li><a href="2010/04">April 2010</a></li>
      <li><a href="2010/03">March 2010</a></li>
      <li><a href="2010/02">February 2010</a></li>
      <li><a href="2010/01">January 2010</a></li>
    </ul>
  </nav>
</section>
```

Определение структуры блога на этом завершается. Теперь можно переходить к применению стилей к новым элементам.

Стилевое оформление

Стилевое оформление применяется к новым элементам точно так же, как к тегам `div`. Сначала мы создаем новый файл таблицы стилей *style.css* и присоединяем его к документу HTML, включая ссылку в заголовок.

html5newtags/index.html

```
<link rel="stylesheet" href="style.css" type="text/css">
```

Выровняем содержимое страницы по центру, а также зададим базовые стили шрифтов.

html5newtags/style.css

```
body{
  width:960px;
  margin:15px auto;
  font-family: Arial, "MS Trebuchet", sans-serif;
}
```

```
p{
  margin:0 0 20px 0;
}

p, li{
  line-height:20px;
}
```

Затем определим ширину заголовка.

html5newtags/style.css

```
header#page_header{
  width:100%;
}
```

Стилевое оформление навигационных ссылок осуществляется преобразованием маркированных списков в горизонтальные навигационные панели.

html5newtags/style.css

```
header#page_header nav ul, #page_footer nav ul{
  list-style: none;
  margin: 0;
  padding: 0;
}
#page_header nav ul li, footer#page_footer nav ul li{
  padding:0;
  margin: 0 20px 0 0;
  display:inline;
}
```

Раздел posts размещается слева, а также задается его ширина; выноска в статье размещается справа. Раз уж мы занимаемся оформлением выноски, заодно увеличим размер шрифта.

html5newtags/style.css

```
section#posts{
  float: left;
  width: 74%;
}

section#posts aside{
  float: right;
  width: 35%;
```

html5newtags/index.html

```
<!--[if lt IE 9]>
<script type="text/javascript">
  document.createElement("nav");
  document.createElement("header");
  document.createElement("footer");
  document.createElement("section");
  document.createElement("aside");
  document.createElement("article");
</script>
<![endif]-->
```

Этот конкретный комментарий предназначен для любой версии Internet Explorer ранее 9.0. Если перезагрузить страницу, она будет выглядеть правильно.

Однако следует учитывать, что при этом работоспособность страницы начинает зависеть от JavaScript. Улучшение структуры и удобочитаемости того стоит: проблем с доступностью нет, потому что контент будет нормально воспроизводиться и читаться экранным диктором. В итоге вы рискуете только тем, что страница будет выглядеть безнадежно устаревшей в браузерах пользователей, намеренно отключивших JavaScript.

Представленное решение хорошо подходит как для добавления поддержки небольшого количества элементов, так и для изучения самой возможности добавления поддержки. Замечательный проект Реми Шарпа HTMLshiv[1] развивает эту идею; вероятно, он лучше подойдет для добавления обходной поддержки при намного большем количестве поддерживаемых элементов.

[1] http://code.google.com/p/html5shiv/

```
html5_popups_with_custom_data/original_example_1.html
<a href='#'
  onclick="window.open(holiday_pay.html',WinName,'width=300,heig
  ht=300);">
  Holiday pay
</a>
```

Такой способ создания ссылок, открывающих всплывающие окна, весьма широко распространен. Собственно, многие начинающие программисты на JavaScript учатся создавать всплывающие окна именно таким образом. Однако у этого способа имеются свои недостатки, которые необходимо упомянуть, прежде чем двигаться дальше.

Улучшение доступности

В ссылке не задан адрес места назначения! Если пользователь отключит JavaScript, то ссылка не приведет его на нужную страницу. Это серьезная проблема, которую необходимо решать немедленно. *Никогда* не опускайте атрибут `href` и не присваивайте ему фиктивные значения — *ни при каких условиях*. Присвойте адрес ресурса, который должен открываться во всплывающем окне.

```
Download html5_popups_with_custom_data/original_example_2.html
<a href='holiday_pay.html'
onclick="window.open(this.href,WinName,'width=300,height=300);">
  Holiday pay
</a>
```

Код JavaScript читает адрес ссылки из атрибута `href` присоединенного элемента.

Чтобы ваши страницы обладали хорошей доступностью, прежде всего убедитесь в том, что вся функциональность работает без JavaScript.

Отказ от onclick

Поведение должно быть отделено от контента — по аналогии с тем, как информация представления хранится отдельно от контента в таблицах стилей. Поначалу решение с `onclick` кажется удобным, но представьте страницу с пятьюдесятью ссылками, и вы увидите, что решения с методом `onclick` быстро выходят из-под контроля. Один и тот же код

JavaScript будет повторяться снова и снова. А если этот код генерируется каким-то серверным кодом, то количество событий JavaScript возрастает, а размер итогового кода HTML намного больше необходимого.

Вместо этого назначьте каждому якорю на странице идентифицирующий класс.

html5_popups_with_custom_data/original_example_3.html

```
<a href="holiday_pay" class="popup">Holiday Pay</a>
```

html5_popups_with_custom_data/original_example_3.html

```
var links = $("a.popup");

links.click(function(event){
    event.preventDefault();
    window.open($(this).attr('href'));
});
```

Мы используем селектор jQuery для получения элемента с классом popup, после чего добавляем наблюдателя для события click. Код, передаваемый методу click, будет выполняться каждый раз, когда пользователь щелкает на ссылке. Метод preventDefault отменяет стандартное поведение события щелчка. В данном случае отменяется переход браузера по ссылке и вывод новой страницы.

Однако по сравнению с исходным примером мы кое-что потеряли — информацию о размерах и местоположении окна, присутствовавшую в исходном примере. Дизайнер страницы, не владеющий JavaScript, должен иметь возможность задать размеры окна на уровне ссылки.

Нам помогут пользовательские атрибуты данных!

Подобные ситуации часто встречаются при создании любых приложений с использованием JavaScript. Как мы уже видели, высоту и ширину окна желательно хранить в коде, но решение с onclick имеет много недостатков. Тем не менее эти данные можно встроить в атрибуты элемента. Для этого ссылка должна строиться следующим образом.

html5_popups_with_custom_data/popup.html

```
a href="help/holiday_pay.html"
    data-width="600"
```

```
    data-height="400"
    title="Holiday Pay"
    class="popup">Holiday pay</a>
```

Теперь остается лишь изменить написанное нами событие `click`, чтобы оно получало значения из пользовательских атрибутов данных ссылки и передавало их методу `window.open`.

html5_popups_with_custom_data/popup.html

```
$(function(){
  $(".popup").click(function(event){
    event.preventDefault();
    var href = $(this).attr("href");
    var width = $(this).attr("data-width");
    var height = $(this).attr("data-height");
    var popup = window.open (href,"popup",
        "height=" + height +",width=" + width + "");
  });
});
```

Вот и все! Ссылка открывается в новом окне.

ПРЕДУПРЕЖДЕНИЕ

В этом примере в пользовательских атрибутах данных сохранялась дополнительная информация, используемая сценарием клиентской стороны. Это умное решение конкретной проблемы демонстрирует один из способов использования атрибутов. Хотя в нем информация представления смешивается с разметкой, этот простой способ наглядно показывает, как легко читаются значения, встроенные в страницу, из кода JavaScript.

Обходное решение

Атрибуты уже сейчас работают в старых браузерах (при условии, что они поддерживают JavaScript). Пользовательские атрибуты данных не помешают работе браузера, а документ пройдет проверку валидности для `doctype` HTML5, так как атрибуты, имена которых начинаются с `data-`, игнорируются.

Перспективы

Со временем новые теги и атрибуты получат более широкую поддержку, и перед вами откроется немало интересных возможностей. Например,

можно будет легко идентифицировать и отключать области навигации и завершители статей при помощи стилевых таблиц печати.

```
nav, article>footer{display:none}
```

При помощи языка сценариев можно быстро идентифицировать все статьи на странице или на сайте. Но самое важное, что контент помечается описывающими его тегами — это способствует улучшению таблиц стилей и кода JavaScript.

Пользовательские атрибуты данных обладают гибкостью, необходимой для включения в разметку произвольной информации. В главе 6 мы еще вернемся к их использованию. Например, их можно использовать в JavaScript для определения того, должен ли тег формы осуществлять отправку данных через Ajax — для этого достаточно найти тег `form` с атрибутом `data-remote=true` (как делает среда разработки Ruby on Rails).

Возможности использования атрибутов весьма разнообразны: скажем, ими можно воспользоваться для отображения даты и времени в часовом поясе пользователя при кэшировании страницы. Просто сохраните дату на HTML-странице в формате UTC и преобразуйте ее в местное время на стороне клиента. Пользовательские атрибуты позволяют встраивать в страницы реальные, действительно нужные данные; сейчас все больше библиотек и инфраструктур активно использует их. Безусловно, вы найдете им полезное применение в своей работе.

И эпидемия дивита будет побеждена раз и навсегда!

3 Новые возможности веб-форм

Все разработчики, занимавшиеся созданием сложных пользовательских интерфейсов, отлично знают, что возможности элементов форм HTML сильно ограничены. Текстовые поля, меню, переключатели, флажки — да еще неуклюжие списки с множественным выделением, работу с которыми приходится дополнительно разъяснять пользователям («Удерживая клавишу Ctrl, щелкайте на нужных строках, а если вы работаете на Mac, то используйте клавишу Cmd»).

И тогда вы делаете то, что делают все нормальные люди, — используете Prototype или jQuery или реализуете собственные элементы или функции в виде комбинации HTML, CSS и JavaScript. А потом, взглянув на форму с множеством ползунков, календарей, счетчиков, полей с заполняющим текстом и визуальных редакторов, вы быстро понимаете, что создали сущий кошмар для себя. Вам придется следить за тем, чтобы элементы, добавленные на страницу, не конфликтовали с другими элементами или библиотеками JavaScript на странице. Вы можете потратить несколько часов на собственную реализацию календаря, а потом обнаружить, что она конфликтует с библиотекой Prototype, потому что jQuery перехватывает функцию `$()`. Тогда вы используете метод jQuery() `noConflict()`, но из-за этого перестает работать элемент выбора цвета, потому что плагин был написан недостаточно аккуратно.

Улыбнулись? Наверное, потому, что описанная ситуация вам хорошо знакома. Нахмурились? Видимо, по той же причине. Однако не стоит отчаиваться. В этой главе мы построим пару веб-форм с полями новых типов, а также реализуем автофокус и заполняющий текст.

В завершение мы рассмотрим, как с помощью нового атрибута `contenteditable` преобразовать любое поле HTML в редактируемый элемент.

Конкретно в этой главе рассматриваются следующие возможности[1]:

`<input type="email">`

Поле для ввода адреса электронной почты. [O10.1, IOS]

`<input type="url">`

Поле для ввода URL-адреса. [O10.1, IOS]

`<input type="tel">`

Поле для ввода телефонного номера. [O10.1, IOS]

`<input type="search">`

Поле для ввода ключевых слов поиска. [C5, S4, O10.1, IOS]

`<input type="range">`

Ползунок. [C5, S4, O10.1]

`<input type="number">`

Поле для ввода числовых данных, часто в виде элемента-счетчика. [C5, S5, O10.1, IOS]

`<input type="date">`

Поле для ввода даты. Поддерживаются типы `date`, `month` и `week`. [C5, S5, O10.1]

`<input type="datetime">`

Поле для ввода даты со временем. Поддерживаются типы `datetime`, `datetime-local` и `time`. [C5, S5, O10.1]

`<input type="color">`

Поле для выбора цвета. [C5, S5] (Chrome 5 и Safari 5 «понимают» поле `Color`, но не отображают конкретный элемент.)

`<input type="text" autofocus>`

Поддержка передачи фокуса конкретному элементу формы. [C5, S4]

`<input type="email" placeholder="me@example.com">`

---

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

Поддержка автоматического включения текста в поле формы. [C5, S4, F4]

```
<p contenteditable>lorem ipsum</p>
```

Поддержка редактирования контента «на месте» в браузере. [C4, S3.2, IE6, O10.1]

Итак, для начала познакомимся поближе с некоторыми из этих исключительно полезных типов полей.

Рецепт 3. Описание данных при помощи новых полей

В HTML5 появились новые типы полей форм, позволяющие более точно описать вводимые пользователем данные. В дополнение к стандартным элементам (текстовые поля, переключатели и флажки), формы также могут содержать такие элементы, как поля адресов электронной почты, календари, палитры для выбора цвета, счетчики (spinboxes) и ползунки. Браузеры сами отображают расширенные элементы, а разработчик избавляется от необходимости использовать JavaScript. На мобильных устройствах, а также виртуальных клавиатурах планшетных и сенсорных устройств тип поля может использоваться для выбора раскладки клавиатуры. Например, на iPhone при вводе данных в полях `URL` и `email` браузер Mobile Safari отображает альтернативную раскладку с удобным вводом специальных символов `@`, `.`, `:` и `/`.

Улучшение формы проекта AwesomeCo

Фирма AwesomeCo работает над новым веб-приложением, предназначенным для управления проектами. В этом приложении разработчики и руководители смогут отслеживать ход работы по разным проектам, которыми они занимаются. Для каждого проекта указано название, контактный адрес электронной почты и URL-адрес, по которому руководители могут просмотреть предварительную версию создаваемого приложения. Также на форме должны отображаться поля для начальной даты, приоритета и приблизительного количества рабочих часов до завершения проекта. Наконец, начальник отдела разработки хочет назначить каждому проекту определенный цвет, чтобы ему было проще опознавать проекты при просмотре отчетов.

Давайте построим макет страницы описания проекта с использованием новых полей форм HTML5.

Построение базовой формы

Давайте построим базовую форму HTML, которая осуществляет отправку данных методом POST. Поле названия проекта ничем особенным не отличается, поэтому мы воспользуемся старым и проверенным типом `text`.

html5forms/index.html

```
<form method="post" action="/projects/1">

  <fieldset id="personal_information">
    <legend>Project Information</legend>
    <ol>
      <li>
        <label for="name">Name</label>
        <input type="text" name="name" autofocus id="name">
      </li>
      <li>
        <input type="submit" value="Submit">

      </li>
    </ol>

  </fieldset>

</form>
```

Обратите внимание: при создании разметки формы в упорядоченный список включаются метки (элементы `label`). Они играют важную роль в обеспечении доступности форм. Атрибут `for` ссылается на идентификатор ассоциированного элемента формы; это помогает экранным дикторам идентифицировать поля на странице. Упорядоченный список является хорошим средством перечисления полей без использования сложных таблиц или структур `div`. Кроме того, он позволяет задать порядок, в котором, по вашему мнению, должны заполняться поля.

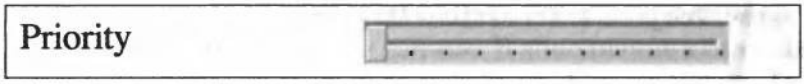
Создание ползунка

Ползунки (sliders) используются для увеличения или уменьшения числовых значений. В нашем приложении этот элемент хорошо подойдет для визуального представления и изменения приоритета проекта. Ползунок реализуется полем типа `range`.

html5forms/index.html

```
<label for="priority">Priority</label>
<input type="range" min="0" max="10"
    name="priority" value="0" id="priority">
```

Добавьте его на форму в новом элементе `li` (по аналогии с предыдущим полем). В Chrome и Opera реализован виджет Slider, который выглядит так:



Также обратите внимание на определение нижней и верхней границ диапазона `min` и `max`. Они ограничивают значения, которые могут храниться в поле формы.

Счетчики и работа с числовыми данными

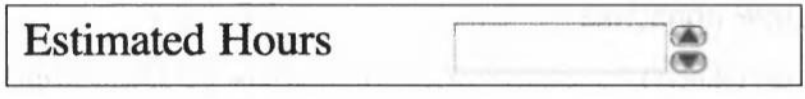
Мы часто работаем с числовыми данными. И хотя ввести число с клавиатуры несложно, счетчики (spinboxes) удобны для незначительного изменения числовых данных. Счетчик представляет собой поле, справа от которого находятся две кнопки со стрелками — для увеличения и уменьшения хранимого значения.

На нашей форме в поле счетчика будет храниться приблизительный объем работы в часах. Это позволит легко изменить значение в случае необходимости.

html5forms/index.html

```
<label for="estimated_hours">Estimated Hours</label>
<input type="number" name="estimated_hours"
min="0" max="1000"
id="estimated_hours">
```

В Opera поддерживается элемент-счетчик, который выглядит так:



В поле счетчика по умолчанию разрешен прямой ввод с клавиатуры. Как и в случае с ползунками, для поля можно задать минимальное и максимальное значение. Однако эти значения не распространяются на значения, непосредственно введенные в поле.

Также обратите внимание на возможность управления приращением, которое определяется параметром `step`. По умолчанию приращение равно 1, но ему также можно присвоить любое числовое значение.

Дата

Дата начала работы над проектом весьма важна, поэтому работа с ней должна быть по возможности удобной. Для хранения даты идеально подходит поле типа `date`.

`html5forms/index.html`

```
<label for="start_date">Start date</label>
<input type="date" name="start_date" id="start_date"
    value="2010-12-01">
```

На момент написания книги полноценный календарный элемент для выбора даты поддерживался только в Opera.

Пример реализации.

Safari 5.0 отображает поле, похожее на числовое, но со стрелками для увеличения и уменьшения даты. Если поле не заполнено, по умолчанию используется значение `"1582"`. Другие браузеры используют в качестве реализации обычное текстовое поле.

Адрес электронной почты

В спецификации HTML5 сказано, что поле типа `email` предназначено для хранения либо отдельного адреса, либо списка адресов электронной почты. Таким образом, это поле идеально подходит для нашего приложения.

`html5forms/index.html`

```
<label for="email">Email contact</label>
<input type="email" name="email" id="email">
```

Поля этого типа приносят особенно заметную пользу на мобильных устройствах, на которых раскладка виртуальной клавиатуры изменяется для удобства ввода адресов электронной почты.

URL-адрес

Также существует особый тип полей для хранения URL-адресов. Этот тип особенно удобен, если ваши посетители просматривают сайт с iPhone — устройство отображает совершенно иную раскладку клавиатуры с кнопками для ускоренного ввода веб-адресов (по аналогии с клавиатурой, отображаемой при вводе URL-адреса в адресной строке Mobile Safari). Чтобы включить в форму поле для URL-адреса, достаточно добавить в нее следующий фрагмент:

html5forms/index.html

```
<label for="url">Staging URL</label>
<input type="url" name="url" id="url">
```

Виртуальные клавиатуры также отображают другую раскладку для полей этого типа.

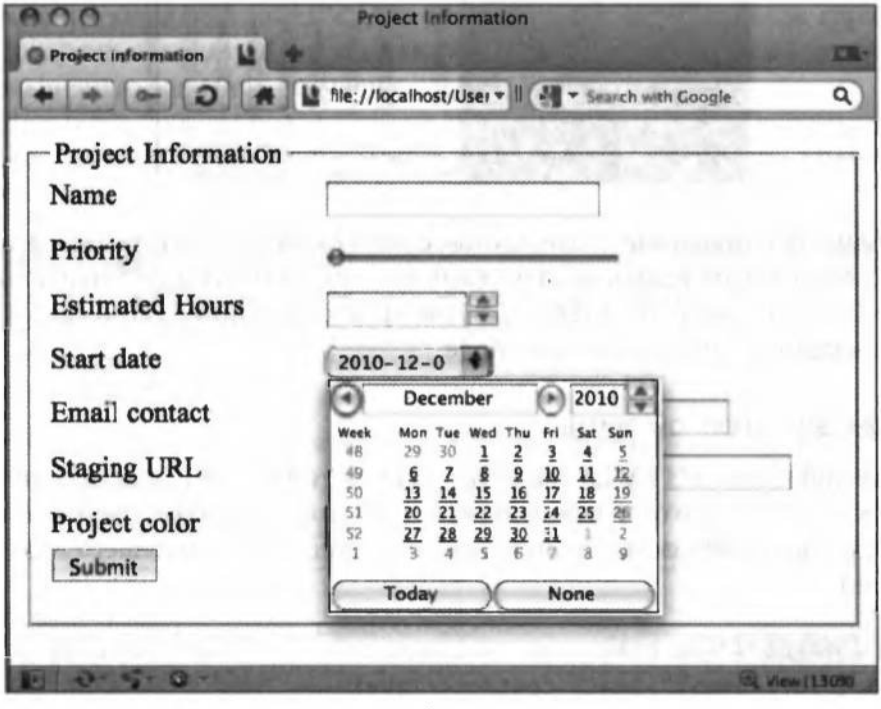


Рис. 3.1. Некоторые элементы форм уже поддерживаются в Opera

Цвет

Наконец пользователю необходимо дать возможность выбрать цветовой код проекта. Для этой цели будет использовано поле типа `color`.

html5forms/index.html

```
<label for="project_color">Project color</label>
<input type="color" name="project_color" id="project_color">
```

На момент написания книги ни один браузер не отображал элемент выбора цвета, но это не препятствовало использованию полей в страницах. Использование правильной разметки для описания контента пригодится в будущем, особенно если потребуется предоставить обходную поддержку.

Opera сейчас поддерживает большинство новых элементов (рис. 3.1), но при открытии страницы в Firefox, Safari или Google Chrome вы не увидите никаких изменений. С этим нужно что-то делать.

Обходное решение

Браузеры, в которых новые типы полей не реализованы, просто заменяют их текстовыми полями, чтобы с формой по крайней мере можно было работать. К полю можно привязать один из виджетов jQuery UI или YUI. Со временем эти элементы будут поддерживаться всеми основными браузерами, и тогда перехватчики JavaScript можно будет удалить.

Замена элемента выбора цвета

Элемент выбора цвета легко идентифицируется и заменяется средствами jQuery с селекторами атрибутов CSS3. Мы ищем любое поле `input` с типом `color` и применяем плагин jQuery с именем SimpleColor.

html5forms/index.html

```
if (!hasColorSupport()){
  $('input[type=color]').simpleColor();
}
```

Так как в разметке используются новые типы форм, добавлять имя класса или другую разметку для идентификации элементов выбора цвета не нужно. Селекторы атрибутов хорошо сочетаются с HTML5.

Если браузер имеет встроенную поддержку элемента выбора цвета, то использовать плагин не нужно, поэтому мы включим фрагмент кода JavaScript для проверки того, поддерживает ли браузер поле `input` с типом `color`.

html5forms/index.html

```
1 function hasColorSupport(){
-   input = document.createElement("input");
-   input.setAttribute("type", "color");
-   var hasColorType = (input.type !== "text");
5   // Проверка частичной реализации Safari/Chrome
-   if(hasColorType){
-     var testString = "foo";
-     input.value=testString;
-     hasColorType = (input.value != testString);
10  }
-   return(hasColorType);
- }
```

Сначала мы в обычном коде JavaScript создаем элемент и присваиваем его атрибуту `type` значение `color`. Затем чтение атрибута `type` проверяет, разрешил ли браузер задать значение атрибута. Если чтение возвращает значение `color`, значит, этот тип элемента поддерживается, а если нет — нужно применять сценарий.

В строке 6 начинается самое интересное. В Safari 5 и Google Chrome 5 тип `color` реализован частично: браузеры поддерживают поле, но не отображают виджет. На странице все равно создается текстовое поле. Соответственно в своем проверочном методе мы задаем значение поля и проверяем, сохранилось ли оно. Если значение не сохранилось, предполагается, что браузер реализует выбор цвета, потому что элемент ведет себя не как текстовое поле.

Код, заменяющий элемент выбора цвета, выглядит так:

html5forms/index.html

```
if (!hasColorSupport()){
  $('input[type=color]').simpleColor();
}
```

Это решение работает, но оно крайне ненадежно: оно работает только с конкретными браузерами и только для элемента выбора цвета. У других элементов имеются свои особенности, которые вам тоже придется изучать. К счастью, существует альтернативное решение.

Modernizr

Библиотека Modernizr[1] обнаруживает поддержку многих возможностей HTML5 и CSS3. Она не реализует отсутствующую функциональность, но предоставляет ряд надежных механизмов проверки полей форм, сходных с использованным в нашем решении, но более надежных.

Прежде чем включать Modernizr в свои проекты, обязательно разберитесь в том, как работает библиотека. Если вы используете код в своем проекте, то отвечаете за него именно вы, кто бы ни был автором этого кода — вы или кто-то другой. В текущей версии библиотека Modernizr не справлялась с частичной поддержкой элемента выбора цвета в Safari. Возможно, с выходом следующей версии Chrome или Firefox вам придется создавать собственное решение для преодоления подобных проблем — и может быть, вы предоставите это решение для использования в Modernizr!

Обходные решения для элементов выбора даты, ползунков и так далее реализуются аналогичным образом. Ползунки и элементы выбора даты включены в виде компонентов в библиотеку jQuery UI[2]. Включите библиотеку jQuery UI в страницу, проверьте, реализована ли в браузере встроенная поддержка элемента, и если нет — примените JavaScript-версию.

С течением времени вы сможете полностью исключить элементы JavaScript и положиться только на поддержку элементов браузером. Из-за сложностей, возникающих при проверке этих типов, библиотека Modernizr оказывается очень полезной. Тем не менее в оставшейся части книги мы будем писать собственные методы проверки, чтобы вы видели, как они работают.

Кроме новых типов полей форм, в HTML5 также появились новые атрибуты для улучшения доступности страниц. Давайте посмотрим, как использовать поддержку автофокуса.

---

[1] http://www.modernizr.com/

[2] http://jqueryui.com/

Рецепт 4. Использование автофокуса для перехода к первому полю

Чтобы значительно ускорить ввод данных, установите курсор в первое поле формы при загрузке страницы. Многие поисковые системы делают это средствами JavaScript, а теперь спецификация HTML5 предоставляет такую возможность на уровне самого языка.

Все, что от вас потребуется — это добавить атрибут `autofocus` к любому полю формы, как было сделано ранее при построении базовой формы (с. 50).

html5forms/index.html

```
<label for="name">Name</label>
<input type="text" name="name" autofocus id="name">
```

Чтобы механизм автофокуса работал надежно, на странице должен быть только один атрибут `autofocus`. Если таких элементов будет несколько, браузер размещает курсор в последнем поле автофокуса.

Обходное решение

Если браузер пользователя не поддерживает автофокус, найдите атрибут `autofocus` в коде JavaScript, а затем используйте jQuery для передачи фокуса элементу. Вероятно, это самое простое обходное решение из всех приведенных в книге.

html5forms/autofocus.js

```
function hasAutofocus() {
  var element = document.createElement('input');
  return 'autofocus' in element;
}

$(function(){
  if(!hasAutofocus()){
    $('input[autofocus=true]').focus();
  }
});
```

Просто включите этот фрагмент JavaScript в свою страницу, — и поддержка автофокуса появится там, где она вам нужна.

Автофокус немного упрощает начало работы с формой после ее загрузки. Но чтобы форма стала еще более понятной, предоставьте пользователям более подробную информацию о данных, которые должны вводиться в полях. Перейдем к рассмотрению атрибута `placeholder`.

Рецепт 5. Заполняющий текст

Заполняющий текст сообщает пользователям, как следует вводить данные в поле формы. Пример формы с заполняющим текстом показан на рис. 3.2. Давайте посмотрим, как строится такая форма.

Рис. 3.2. Заполняющий текст подсказывает, какую информацию следует ввести в поле

Простая форма регистрации новых работников

На вспомогательном сайте AwesomeCo пользователи вводят информацию о себе для создания учетной записи. Одна из основных проблем с регистрацией заключается в том, что пользователи постоянно пытаются выбирать ненадежные пароли. При помощи заполняющего текста мы дадим пользователю некоторое представление о действующих требованиях к паролю. Ради единства оформления заполняющий текст также включается и в другие поля.

Чтобы включить в поле заполняющий текст, добавьте атрибут `placeholder`:

`html5placeholdertext/index.html`

```
<input id="email" type="email"
name="email" placeholder="user@example.com">
```

Вся разметка формы с заполняющим текстом для всех полей выглядит примерно так.

html5placeholdertext/index.html

```
<form id="create_account" action="/signup" method="post">
  <fieldset id="signup">
    <legend>Create New Account</legend>
    <ol>
      <li>
        <label for="first_name">First Name</label>
        <input id="first_name" type="text"
               autofocus="true"
               name="first_name" placeholder="'John'">
      </li>
      <li>
        <label for="last_name">Last Name</label>
        <input id="last_name" type="text"
               name="last_name" placeholder="'Smith'">
      </li>
      <li>
        <label for="email">Email</label>
        <input id="email" type="email"
               name="email" placeholder="user@example.com">
      </li>
      <li>
        <label for="password">Password</label>
        <input id="password" type="password" name="password"
        value=""
               autocomplete="off" placeholder="8-10
               characters" />
      </li>
      <li>
        <label for="password_confirmation">Password Confirmation</
        label>
        <input id="password_confirmation" type="password"
               name="password_confirmation" value=""
               autocomplete="off" placeholder="Type your
               password again" />
      </li>
      <li><input type="submit" value="Sign Up"></li>
    </ol>
  </fieldset>
</form>
```

Запрет автозаполнения

Возможно, вы заметили, что в поле `password` этой формы добавлен атрибут `autocomplete`. В HTML5 появился атрибут `autocomplete`, который сообщает браузерам, что они не должны автоматически заполнять поле данными. Некоторые браузеры запоминают данные, ранее вводившиеся пользователем; в некоторых ситуациях следует указать браузеру, что делать этого не стоит.

Так как для хранения полей формы снова используется упорядоченный список, мы добавим немного простейшего кода CSS для улучшения внешнего вида формы.

html5placeholdertext/style.css

```
fieldset{
  width: 216px;
}

fieldset ol{
  list-style: none;
  padding:0;
  margin:2px;
}

fieldset ol li{
  margin:0 0 9px 0;
  padding:0;
}

/* Поля размещаются в отдельных строках */
fieldset input{
  display:block;
}
```

Теперь пользователи Safari, Opera и Chrome увидят в полях формы содержательный текст. Остается реализовать аналогичную функциональность в Firefox и Internet Explorer.

Обходное решение

Код JavaScript позволяет включить заполняющий текст в поля формы без особых усилий. Проверьте значение каждого поля формы, и если оно пустое — задайте `value` заполняющий текст. При получении фокуса значение сбрасывается, а при потере фокуса проверяется заново. Если

содержимое поля изменилось, вы его не трогаете, а если поле пустое — в него помещается заполняющий текст.

Поддержка заполняющего текста проверяется так же, как и поддержка автофокуса.

html5placeholdertext/index.html

```
function hasPlaceholderSupport() {
  var i = document.createElement('input');
  return 'placeholder' in i;
}
```

Далее пишется код JavaScript для обработки изменений. Наше решение базируется на работе Эндрю Дженуари[1] и др. Все поля формы заполняются текстом, хранимым в атрибуте `placeholder`. Когда пользователь выделяет поле, мы удаляем заполняющий текст, помещенный в поле. Код упакован в плагин jQuery, чтобы его поведение было проще применять к нашей форме. О том, как работают плагины, рассказано во врезке на с. 66.

html5placeholdertext/jquery.placeholder.js

```
(function($){

  $.fn.placeholder = function(){

    function valueIsPlaceholder(input){
      return ($(input).val() == $(input).
      attr("placeholder"));
    }
    return this.each(function() {

      $(this).find(":input").each(function(){

        if($(this).attr("type") == "password"){

          var new_field = $("<input type='text'>");
          new_field.attr("rel", $(this).attr("id"));
          new_field.attr("value", $(this).
          attr("placeholder"));
          $(this).parent().append(new_field);
```

[1] Исходный сценарий доступен по адресу http://www.morethannothing.co.uk/wp-content/uploads/2010/01/placeholder.js, но он не поддерживает поля паролей в IE.

```
-           new_field.hide();
-
20          function showPasswordPlaceHolder(input){
-             if( $(input).val() == "" ||
              valueIsPlaceholder(input) ){
-               $(input).hide();
-               $('input[rel=' + $(input).attr("id") + ']').
                show();
-             };
25          };
-
-           new_field.focus(function(){
-             $(this).hide();
-             $('input#' + $(this).attr("rel")).show().focus();
30          });
-
-           $(this).blur(function(){
-             showPasswordPlaceHolder(this, false);
-           });
35
-           showPasswordPlaceHolder(this);
-
-         }else{
-
40          // Значение заменяется заполняющим текстом.
-           // Необязательный параметр reload решает проблему
-           // кэширования полей в FF и IE.
-           function showPlaceholder(input, reload){
-             if( $(input).val() == "" ||
45             ( reload && valueIsPlaceholder(input) ) ){
-               $(input).val($(input).attr("placeholder"));
-             }
-           };
-
50          $(this).focus(function(){
-             if($(this).val() == $(this).attr("placeholder")){
-               $(this).val("");
-             };
-           });
55
-           $(this).blur(function(){
-             showPlaceholder($(this), false)
-           });
```

```
-
60
-             showPlaceholder(this, true);
-           };
-         });
-
65       // Заполняющий текст не должен отправляться Формой
-        $(this).submit(function(){
-          $(this).find(":input").each(function(){
-            if($(this).val() == $(this).attr("placeholder")){
-              $(this).val("");
70            }
-           });
-         });
-
-       });
75   };
-
-  })(jQuery);
```

В этом плагине есть пара моментов, заслуживающих внимания. В строке 45 заполняющий текст заносится в поля в том случае, если они не содержат текущего значения, а также при обновлении страницы. Firefox и другие браузеры сохраняют данные, введенные в формах. Мы присваиваем атрибуту `value` заполняющий текст, но, разумеется, не хотим, чтобы он случайно превратился в фактические данные, введенные пользователями. При загрузке страницы методу передается значение `true` (строка 61).

Поля паролей по своему поведению несколько отличаются от других полей форм, поэтому их приходится обрабатывать особым образом. Взгляните на строку 12: мы обнаруживаем поле пароля и заменяем его обычным текстовым полем, чтобы значение не маскировалось звездочками. Некоторые браузеры выдают ошибку при попытке преобразования типа поля, содержащего пароль, поэтому нам приходится заменять поле с паролем текстовым полем.

Этот прием изменяет текущие значения полей формы; вероятно, заполняющий текст предназначен только для пользователя и не должен передаваться на сервер. Так как обходное решение работает только при включенном коде JavaScript, мы можем воспользоваться JavaScript для анализа отправленных данных и удаления всех значений, совпадающих с заполняющим текстом (строка 66).

ПЛАГИНЫ JQUERY

Функциональность jQuery расширяется при помощи плагинов. Вы добавляете собственные методы в функцию jQuery, и ваш плагин становится доступным для любого разработчика, включившего вашу библиотеку. Рассмотрим тривиальный пример с вызовом окна сообщения JavaScript.

```
jQuery.fn.debug = function() {
  return this.each(function(){
    alert(this.html());
  });
```

Чтобы окно отображалось для каждого абзаца на странице, достаточно вызвать метод следующим образом:

```
$("p").debug();
```

Плагины jQuery перебирают и возвращают коллекции объектов jQuery, поэтому их можно объединять в цепочки. Например, поскольку наш плагин debug также возвращает коллекцию jQuery, мы можем воспользоваться методом jQuery css для изменения цвета текста этих абзацев — и все это в одной строке.

```
$("p").debug().css("color", "red");
```

Плагины jQuery неоднократно используются в книге для организации кода при создании обходных решений. За дополнительной информацией обращайтесь на сайт с документацией jQuery[1].

После того как код будет оформлен в виде плагина, чтобы активизировать его для страницы, достаточно присоединить его к форме.

html5placeholdertext/index.html

```
$(function(){
  function hasPlaceholderSupport() {
    var i = document.createElement('input');
    return 'placeholder' in i;
  }

  if(!hasPlaceholderSupport()){
    $("#create_account").placeholder();
    //END placeholder_fallback

    $('input[autofocus=true]').focus();

  };
});
```

Итак, мы построили вполне достойное решение, которое позволяет использовать заполняющий текст в веб-приложении независимо от типа браузера.

[1] http://docs.jquery.com/Plugins/Authoring

Рецепт 6. Редактирование «на месте»

Веб-разработчик всегда старается по возможности упростить работу со своими приложениями. Иногда бывает удобно, чтобы пользователь сайта мог быстро отредактировать информацию о себе без перехода к другой форме. Традиционная реализация редактирования «на месте» основана на отслеживании текстовых областей, в которых делаются щелчки, и замене этих областей текстовыми полями. Поля отправляют измененный текст серверу средствами Ajax. Появившийся в HTML5 тег `contenteditable` автоматически решает проблему ввода данных. Разработчику придется написать код JavaScript для отправки введенных данных на сервер, где эти данные будут сохранены, но по крайней мере ему уже не нужно возиться с созданием и переключением скрытых полей.

В одном из текущих проектов AwesomeCo пользователь может просматривать профиль своей учетной записи. В профиле указано имя, город, штат, почтовый индекс и адрес электронной почты. Давайте реализуем на странице профиля функциональность редактирования «на месте», чтобы полученный интерфейс выглядел так, как показано на рис. 3.3.

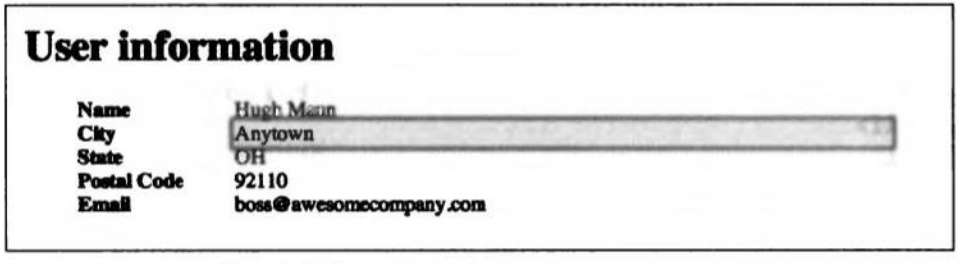


Рис. 3.3. Простое редактирование «на месте»

Прежде чем приступать, я хочу сказать, что реализация некоторой возможности на базе JavaScript без предварительной реализации серверного решения противоречит всем моим убеждениям в области построения доступных веб-приложений. Мы поступим так только потому, что я хочу сосредоточиться на атрибуте `contenteditable`, а наш код *не предназначен для реальной эксплуатации*. Всегда — да, всегда! — начинайте с построения решения, не требующего JavaScript, и только потом переходите к построению версии, основанной на сценарном коде. Наконец, обязательно пишите автоматизированные тесты для обеих ветвей, чтобы повысить вероятность обнаружения ошибок в случае изменения только одной из двух версий.

Форма профиля

В HTML5 появился атрибут `contenteditable`, который поддерживается практически всеми элементами. Простое добавление этого атрибута превращает элемент в поле с возможностью редактирования.

`html5_content_editable/show.html`

```
<h1>User information</h1>
<div id="status"></div>
<ul>
  <li>
    <b>Name</b>
    <span id="name" contenteditable="true">Hugh Mann</span>
  </li>
  <li>
    <b>City</b>
    <span id="city" contenteditable="true">Anytown</span>
  </li>
  <li>
    <b>State</b>
    <span id="state" contenteditable="true">OH</span>
  </li>
  <li>
    <b>Postal Code</b>
    <span id="postal_code" contenteditable="true">92110</span>
  </li>
  <li>
    <b>Email</b>
    <span id="email" contenteditable="true">boss@awesomecompany.
    com</span>
  </li>
</ul>
```

Стилевое оформление поля осуществляется средствами CSS. Мы воспользуемся селекторами CSS3 для идентификации редактируемых полей, чтобы они изменяли цвет при наведении указателя мыши или выделении.

`html5_content_editable/show.html`

```
1  ul{list-style:none;}
-
-  li{clear:both;}
```

```
-
 5  li>b, li>span{
-     display: block;
-     float: left;
-     width: 100px;
-   }
10
-   li>span{
-     width:500px;
-     margin-left: 20px;
-   }
15
-   li>span[contenteditable=true]:hover{
-     background-color: #ffc;
-   }
-
20 li>span[contenteditable=true]:focus{
-     background-color: #ffa;
-     border: 1px shaded #000;
-   }
```

Вот и все, что требуется для реализации интерфейсной части. Пользователи могут легко изменять данные на странице; теперь необходимо обеспечить их сохранение.

Сохранение данных

Хотя пользователи могут изменять данные, эти изменения будут потеряны при обновлении или переходе к другой странице. Нам понадобится механизм отправки данных исполнительной подсистеме; задача легко решается при помощи jQuery. Если вы когда-либо работали с Ajax, ничего нового в происходящем здесь вы не увидите.

html5_content_editable/show.html

```
$(function(){
    var status = $("#status");
    $("span[contenteditable=true]").blur(function(){
      var field = $(this).attr("id");
      var value = $(this).text();
      $.post("http://localhost:4567/users/1",
        field + "=" + value,
```

```
        function(data){
          status.text(data);
        }
      );
    });

});
```

Мы добавляем слушателя событий для каждого элемента `span` на странице, у которого атрибут `contenteditable` равен `true`. Далее остается лишь отправить данные сценарию на стороне сервера.

Обходное решение

Некоторые из использованных нами приемов будут работать не у всех пользователей. Во-первых, появилась нежелательная зависимость от JavaScript для сохранения отредактированных результатов на сервере. Во-вторых, псевдокласс `focus`, используемый для выделения полей при получении ими фокуса, не поддерживается некоторыми версиями IE. Начнем с решения проблемы с функциональностью, а затем перейдем к визуальным эффектам.

Создание страницы редактирования

Вместо того чтобы продумывать различные ситуации, которые могут помешать пользователю использовать наш метод, мы просто предоставим ему возможность перейти к отдельной странице с формой. Конечно, объем кода от этого увеличится, но подумайте, сколько разнообразных проблемных сценариев существует:

- У пользователя отключен JavaScript, и он работает в Internet Explorer 7.
- Браузер пользователя несовместим с HTML5.
- Пользователь работает в новейшей версии Firefox с поддержкой HTML5, но отключает JavaScript просто из-за своей неприязни к JavaScript (это происходит сплошь и рядом... намного чаще, чем можно ожидать).

Разумнее всего создать форму, которая выполняет POST-отправку действию, обрабатывающему обновление Ajax. Как именно это делать — решайте сами, но многие среды позволяют проверить тип запроса по

заголовкам `accept`, чтобы узнать, поступил ли запрос в результате обычной POST-отправки или `XMLHttpRequest`. Если браузер поддерживает `contenteditable` и JavaScript, ссылка на форму будет скрываться.

Создайте новую страницу с именем *edit.html*. Закодируйте стандартную форму, которая отправляет данные тому же действию обновления, что и Ajax-версия.

html5_content_editable/edit.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <title>Editing Profile</title>
    <link href="style.css" rel="stylesheet" media="screen">
  </head>
  <body>
    <form action="/users/1" method="post" accept-
charset="utf-8">
      <fieldset id="your_information">
        <legend>Your Information</legend>
        <ol>
          <li>
            <label for="name">Your Name</label>
            <input type="text" name="name" value="" id="name">
          </li>
          <li>
            <label for="city">City</label>
            <input type="text" name="city" value="" id="city">
          </li>
          <li>
            <label for="state">State</label>
            <input type="text" name="state" value="" id="state">
          </li>
          <li>
            <label for="postal_code">Postal Code</label>
            <input type="text" name="postal_code" value=""
            id="postal_code">
          </li>
          <li>
            <label for="email">Email</label>
            <input type="email" name="email" value=""
            id="email">
```

```
        </li>
      </ol>

    </fieldset>

    <p><input type="submit" value="Save"></p>
  </form>

</body>
</html>
```

Затем включите ссылку на эту страницу в *show.html*.

html5_content_editable/show.html

```
<h1>User information</h1>
<section id="edit_profile_link">
  <p><a href="edit.html">Edit Your Profile</a></p>
</section>
<div id="status"></div>
```

После добавления ссылки остается лишь немного изменить наш сценарий. Ссылка на страницу редактирования должна оставаться скрытой, а поддержка Ajax включается только при поддержке редактирования контента.

html5_content_editable/show.html

```
if(document.getElementById("edit_profile_link").contentEditable
!= null){
```

С этой проверкой наш сценарий приобретает следующий вид:

html5_content_editable/show.html

```
$(function(){
  if(document.getElementById("edit_profile_link").contentEditable
!= null){
    $("#edit_profile_link").hide();
    var status = $("#status");
    $("span[contenteditable=true]").blur(function(){
      var field = $(this).attr("id");
      var value = $(this).text();
      $.post("http://localhost:4567/users/1",
          field + "=" + value,
```

```
            function(data){
              status.text(data);
            }
        );
      });
    }

});
```

После реализации этого решения пользователь получает возможность использовать стандартный интерфейс или более быстрый режим редактирования «на месте». Теперь, когда вы знаете, как реализуется этот интерфейс, помните, что начинать следует с реализации обходного решения. Отсутствие реализации этого обходного решения, в отличие от других, нарушает функциональность приложения.

Перспективы

Если вы разместите на своем сайте элемент выбора даты на базе JavaScript, то пользователям придется разбираться в том, как он работает. Каждый, кто когда-либо покупал в Интернете билеты или бронировал номера в гостинице, хорошо знает, как сильно порой отличаются реализации пользовательских элементов форм на сайтах. Нечто похожее происходит и при работе с банкоматом — различия в интерфейсе нередко замедляют выполнение простейших операций.

Но представьте, что на всех сайтах будет использоваться унифицированное поле `date` HTML5, а браузер будет только создавать интерфейс. На всех сайтах, посещенных пользователем, будут отображаться совершенно одинаковые элементы выбора даты. Экранные дикторы даже смогут реализовать стандартный механизм удобного выбора даты для пользователей с ограниченными возможностями по зрению. Теперь подумайте, какую пользу принесет повсеместное использование заполняющего текста и автофокуса. Заполняющий текст подскажет пользователю экранных дикторов, как работают элементы форм, а автофокус упростит навигацию без использования мыши — эта возможность пригодится не только пользователям с ограниченными возможностями по зрению, но и пользователям с двигательной недостаточностью, у которых работа с мышью вызывает затруднения.

Возможность преобразования любого элемента в область редактирования упрощает правку «на месте», но она также может повлиять на построение интерфейсов систем управления контентом.

Как известно, сутью современных веб-приложений является интерактивность, а формы являются важным аспектом этой интерактивности. Усовершенствования HTML5 предоставляют вам набор средств, которые сделают работу пользователей более простой и удобной.

4 Совершенствование пользовательских интерфейсов средствами CSS3

В течение долгого времени разработчикам для реализации нужных эффектов приходилось действовать в обход CSS. Мы использовали JavaScript или код на стороне сервера для чередования окраски строк таблицы, передачи фокуса или реализации эффекта размывки на формах. Нам приходилось загромождать теги дополнительными атрибутами `class` только для того, чтобы разобраться, к какому из 50 полей формы должно применяться стилевое оформление.

Но эти времена прошли! В CSS3 появились замечательные селекторы, которые делают выполнение подобных операций тривиальным. На всякий случай напомню: селектором называется шаблон, используемый для поиска элементов в документе HTML с целью применения к ним стилевого оформления. В этой главе мы используем новые селекторы для оформления таблицы. Далее рассматривается возможность использования других средств CSS3 для улучшения стилевых таблиц печати и разбиения контента на столбцы.

В этой главе будут рассмотрены следующие возможности CSS[1]:

`:nth-of-type [p:nth-of-type(2n+1){color: red;}]`

Поиск всех *n* элементов определенного типа. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

---

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

`:first-child` `[p:first-child{color:blue;}]`

Поиск первого дочернего элемента. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

`:nth-child` `[p:nth-child(2n+1){color: red;}]`

Поиск заданного дочернего элемента в прямом направлении (от начала к концу). [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

`:last-child` `[p:last-child{color:blue;}]`

Поиск последнего дочернего элемента. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

`:nth-last-child` `[p:nth-last-child(2){color: red;}]`

Поиск заданного дочернего элемента в обратном направлении. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

`:first-of-type` `[p:first-of-type{color:blue;}]`

Поиск первого элемента заданного типа. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

`:last-of-type` `[p:last-of-type{color:blue;}]`

Поиск последнего элемента заданного типа. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

Поддержка столбцов:

```
[#content{ column-count: 2; column-gap: 20px; column-rule: 1px
solid #ddccb5; }]
```

Разбиение области контента на несколько столбцов. [C2, F3.5, S3, O9.5, IOS3, A2]

`:after` `[span.weight:after { content: "lbs"; color: #bbb; }]`

Используется с `content` для вставки контента после заданного элемента. [C2, F3.5, S3, IE8, O9.5, IOS3, A2]

`[media="only all and (max-width: 480)"]`

Применение стилей в зависимости от параметров устройства. [C3, F3.5, S4, IE9, O10.1, IOS3, A2]

Рецепт 7. Стилевое оформление таблиц с использованием псевдоклассов

Псевдоклассы CSS предназначены для выбора элементов на основании информации, не входящей в документ, или информации, которая не может быть выражена обычными селекторами. Вероятно, вы уже использовали псевдоклассы ранее — например, `:hover` для изменения цвета ссылки, когда пользователь наводит на нее указатель мыши. В CSS3 появилось несколько новых псевдоклассов, заметно упрощающих поиск элементов.

Работа со счетами

Фирма AwesomeCo использует систему выставления счетов для продаваемых товаров. Одним из основных рынков AwesomeCo является корпоративная сувенирная продукция — ручки, чашки, футболки и вообще все, на что можно «налепить» фирменный логотип. Вам поручено сделать работу со счетами более простой и удобной. В существующей версии системы разработчики создают стандартную таблицу HTML, примерный вид которой показан на рис. 4.1.

Item	Price	Quantity	Total
Coffee mug	$10.00	5	$50.00
Polo shirt	$20.00	5	$100.00
Red stapler	$9.00	4	$36.00
Subtotal			$186.00
Shipping			$12.00
Total Due			$198.00

Рис. 4.1. Таблица HTML без стилевого оформления в текущей версии системы

Перед вами довольно стандартный счет с ценами, количеством единиц товара, суммами строк, промежуточной суммой и общей суммой заказа. Счет было бы удобнее просматривать, если бы строки были окрашены в разные цвета. Также будет полезно изменить цвет итоговой суммы, чтобы она лучше выделялась на общем фоне.

Ниже приведен код таблицы. Скопируйте его в отдельный файл, чтобы с ним было удобнее работать.

css3advancedselectors/table.html

```
table{
  width: 600px;
  border-collapse: collapse;
}
th, td{
  border: none;
}
```

Также немного изменим оформление заголовка: он будет выводиться белыми буквами на черном фоне.

css3advancedselectors/table.html

```
th{
  background-color: #000;
  color: #fff;
}
```

После применения стиля таблица выглядит так:

Item	Price	Quantity	Total
Coffee mug	$10.00	5	$50.00
Polo shirt	$20.00	5	$100.00
Red stapler	$9.00	4	$36.00
Subtotal			$186.00
Shipping			$12.00
Total Due			$198.00

После удаления сетки и увеличения ширины можно переходить к стилевому оформлению отдельных строк и столбцов с использованием псевдоклассов. Начнем с чередования цвета строк.

Чередование цвета строк (:nth-of-type)

Каждый из нас неоднократно видел таблицы с чередованием цвета строк («зебра»): этот эффект полезен тем, что упрощает просмотр данных по строкам. Стилевое оформление такого рода лучше всего выполняется средствами уровня представления, то есть CSS. Традиционно задача решалась включением в строки таблицы дополнительных имен классов (например, **odd** и **even** для нечетных и четных строк соответственно).

Однако подобное загрязнение разметки таблицы нежелательно, поскольку спецификация HTML5 рекомендует избегать использования имен классов для определения представления. При помощи новых селекторов мы сможем добиться желаемого эффекта без изменения разметки — таким образом, представление будет отделено от контента.

Селектор `nth-of-type` находит каждый *n*-й элемент конкретного типа, определяемый формулой или ключевыми словами. Формулы будут более подробно рассмотрены позднее, а пока разберемся с ключевыми словами, потому что их проще понять.

Чтобы каждая вторая строка таблицы была окрашена в другой цвет, проще всего найти все четные строки таблицы и назначить им другой цвет фона. То же самое делается с нечетными строками. В CSS3 имеются ключевые слова `even` и `odd`, предназначенные именно для таких ситуаций.

`css3advancedselectors/table.html`

```
tr:nth-of-type(even){
  background-color: #F3F3F3;
}
tr:nth-of-type(odd) {
  background-color:#ddd;
}
```

Фактически этот селектор означает: «Найти каждую четную строку таблицы и задать ее цвет. Затем найти каждую нечетную строку таблицы и задать ее цвет». Так «зебровая» окраска таблицы реализуется без использования сценарного кода или дополнительных имен классов в строках.

Очередная версия стилевого оформления таблицы выглядит так:

Item	Price	Quantity	Total
Coffee mug	$10.00	5	$50.00
Polo shirt	$20.00	5	$100.00
Red stapler	$9.00	4	$36.00
Subtotal			$186.00
Shipping			$12.00
Total Due			$198.00

Теперь поработаем над выравниванием столбцов в таблице.

Выравнивание текста столбцов (:nth-child)

По умолчанию текст во всех столбцах таблицы выравнивается по левому краю. Мы выровняем по правому краю все столбцы, кроме первого, — чтобы цена и количество единиц товара лучше читались. Для этого мы воспользуемся селектором `nth-child`, но сначала необходимо узнать, как он работает.

Селектор `nth-child` ищет дочерние элементы заданного элемента; по аналогии с `nth-of-type`, он может использовать ключевые слова или формулу.

Формула определяется в виде `an+b`, где `a` — множитель, а `b` — смещение. Принцип использования формул проще понять в контексте; давайте применим его к контексту таблицы.

Для выбора всех строк таблицы можно воспользоваться селектором вида

```
table tr:nth-child(n)
```

В этом примере не указан ни множитель, ни смещение.

Все строки таблицы, кроме первой (строка с заголовками столбцов), выбираются при помощи селектора со смещением:

```
table tr:nth-child(n+2)
```

А для выбора каждой второй строки таблицы используется множитель `2n`:

```
table tr:nth-child(2n)
```

Каждая третья строка выбирается при помощи множителя `3n`.

Если прибавить к множителю смещение, то поиск будет начинаться не от начала таблицы, а с одной из следующих строк. Следующий селектор находит каждую вторую строку, начиная с четвертой:

```
table tr:nth-child(2n+4)
```

Итак, для выравнивания всех столбцов, *кроме* первого, используется следующая запись:

`css3advancedselectors/table.html`

```
td:nth-child(n+2){
  text-align: right;
}
```

Наша таблица постепенно приобретает все более профессиональный вид.

Item	Price	Quantity	Total
Coffee mug	$10.00	5	$50.00
Polo shirt	$20.00	5	$100.00
Red stapler	$9.00	4	$36.00
Subtotal			$186.00
Shipping			$12.00
Total Due			$198.00

На следующем этапе мы изменим оформление последней строки таблицы.

Выделение последней строки (:last-child)

Таблица уже сейчас смотрится вполне прилично, но начальство требует, чтобы нижняя строка выделялась жирным шрифтом. Для этого мы воспользуемся селектором `last-child`, который находит последний дочерний элемент группы.

Многие веб-разработчики устанавливают нижние поля в абзацах для более равномерного распределения текста по странице. Однако иногда это приводит к образованию нежелательных интервалов в конце группы. Например, если абзацы содержатся в боковой панели или на выноске, нижние поля из последнего абзаца желательно исключить, чтобы он не отделялся от границы области. Селектор `last-child` идеально подходит для отмены полей последнего абзаца.

```
p{ margin-bottom: 20px }
#sidebar p:last-child{ margin-bottom: 0; }
```

Аналогичным образом текст нижней строки выделяется жирным шрифтом.

css3advancedselectors/table.html

```
tr:last-child{
  font-weight: bolder;
}
```

Выделение также применяется и к последнему столбцу таблицы, чтобы суммы строк тоже выделялись на общем фоне:

css3advancedselectors/table.html

```
td:last-child{
  font-weight: bolder;
}
```

Наконец, селектор `last-child` используется для увеличения размера шрифта общей суммы в правом нижнем углу таблицы. Мы находим последний столбец последней строки и изменяем его оформление.

css3advancedselectors/table.html

```
tr:last-child td:last-child{
  font-size:24px;
}
```

Item	Price	Quantity	Total
Coffee mug	$10.00	5	$50.00
Polo shirt	$20.00	5	$100.00
Red stapler	$9.00	4	$36.00
Subtotal			$186.00
Shipping			$12.00
Total Due			$198.00

Работа почти закончена. Осталось лишь кое-что сделать с тремя последними строками таблицы.

Поиск в обратном направлении (:nth-last-child)

Если стоимость доставки снижена под действием скидки, то соответствующая строка таблицы должна выделяться цветом. Для быстрого поиска этой строки удобно использовать селектор `nth-last-child`. Вы уже видели, как селектор `nth-child` и формула `an+b` используются для выбора конкретных дочерних элементов (см. раздел «Выравнивание текста столбцов (:nth-child)» этой главы). Селектор `nth-last-child` работает практически так же, если не считать того, что он перебирает дочерние элементы в обратном порядке, начиная с последнего. Это позволяет легко найти предпоследний элемент группы, что, собственно, и нужно сделать в нашем примере.

Итак, оформление строки со стоимостью доставки может быть изменено следующим кодом:

css3advancedselectors/table.html

```
tr:nth-last-child(2){
  color: green;
}
```

Селектор определяет конкретный дочерний элемент — второй с конца.

В оформление таблицы осталось внести последний штрих. Ранее мы выровняли по правому краю все столбцы, кроме первого. Для строк с описаниями и ценами товаров такое выравнивание естественно, но последние три строки выглядят немного странно. Для них лучше использовать выравнивание по правому краю. Для решения этой задачи мы используем селектор `nth-last-child` с отрицательным множителем и положительным смещением.

css3advancedselectors/table.html

```
tr:nth-last-child(-n+3) td{
  text-align: right;
}
```

Такая формула реализует интервальный выбор. В ней используется смещение 3, а с селектором `nth-last-child` выбирается каждый элемент до заданного смещения. Если бы вместо него использовался селектор `nth-child`, то строки выбирались бы от начала таблицы.

Новое оформление таблицы (рис. 4.2) выглядит намного лучше, причем нам совершенно не пришлось изменять разметку. Многие селекторы, использованные в решении, пока недоступны пользователям Internet Explorer; для них нужно разработать обходное решение.

Item	Price	Quantity	Total
Coffee mug	$10.00	5	$50.00
Polo shirt	$20.00	5	$100.00
Red stapler	$9.00	4	$36.00
		Subtotal	$186.00
		Shipping	$12.00
		Total Due	$198.00

Рис. 4.2. Таблица с чередованием цветов и выравниванием, реализованным исключительно средствами CSS3

Обходное решение

Селекторы, использованные в нашем решении, поддерживаются текущими версиями Opera, Firefox, Safari и Chrome, но в Internet Explorer 8.0 и более ранних версий они игнорируются. Необходимо реализовать обходное решение, причем разработчик сталкивается с принципиальным выбором.

Изменение кода HTML

Самое очевидное решение, которое будет работать во всех браузерах, — модификация базового кода. Ко всем ячейкам таблицы присоединяются классы, а для каждого класса задается базовое оформление CSS. Смешение представления с контентом — самый худший вариант; ведь именно для предотвращения подобной мешанины используется CSS3. В будущем вся эта разметка станет лишней, а ее удаление создаст изрядные трудности.

Использование JavaScript

Библиотека jQuery уже поддерживает большинство селекторов CSS3, поэтому написать нужный метод стилевого оформления таблицы несложно, но существует и более простое решение.

Кейт Кларк написал отличную библиотеку IE-css3[1], которая реализует поддержку селекторов CSS3 в Internet Explorer. Для этого необходимо лишь включить пару сценариев в нашу страницу.

Библиотека IE-CSS3 в своей внутренней реализации может использовать jQuery, Prototype или несколько других библиотек, но я предпочитаю библиотеку DOMAssistant[2], потому что в ней реализована наиболее качественная поддержка псевдоклассов.

Загрузите обе библиотеки и подключите их к своему документу. Так как это решение предназначено только для IE, поместите их в условный комментарий, чтобы библиотеки могли использоваться только пользователями IE.

css3advancedselectors/table.html

```
<!--[if (gte IE 5.5)&(lte IE 8)]>
  <script type="text/javascript"
```

[1] http://www.keithclark.co.uk/labs/ie-css3/

[2] http://www.domassistant.com/

```
        src="js/DOMAssistantCompressed-2.8.js"></script>
  <script type="text/javascript"
        src="js/ie-css3.js"></script>
<![endif]-->
```

Включение этих сценариев в страницу решает все проблемы с оформлением таблицы в Internet Explorer. Результат показан на рис. 4.3.

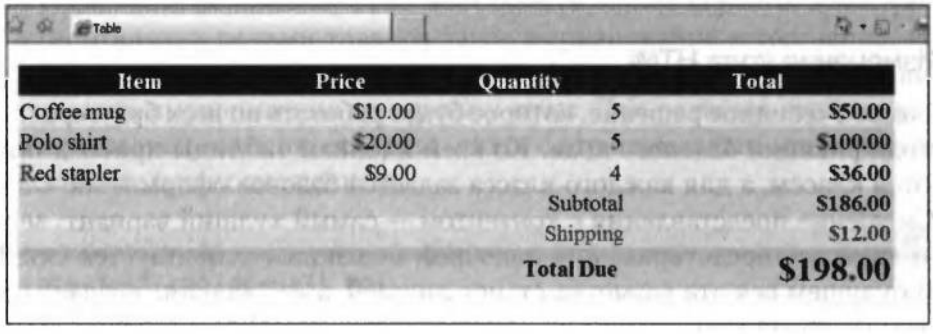


Рис. 4.3. Таблица нормально воспроизводится в Internet Explorer

Решение работает только при включенной поддержке JavaScript, но стилевое оформление предназначено в основном для удобства просмотра. По крайней мере, его отсутствие не помешает пользователю просмотреть содержимое счета.

CSS3 значительно упрощает стилевое оформление элементов, особенно если вы не можете внести изменения в целевой код HTML. Занимаясь стилевым оформлением интерфейсов, по возможности используйте семантическую иерархию и новые селекторы без внесения дополнительной разметки — это существенно упростит сопровождение вашего кода.

Рецепт 8. Печать ссылок (:after)

Технология CSS обычно используется для стилевого оформления существующих документов, но она также позволяет внедрять в документ новый контент. Генерирование контента средствами CSS оправдано в нескольких ситуациях; наиболее очевидный пример — присоединение URL-адреса гиперссылки к ее тексту, когда пользователь печатает страницу. Когда вы просматриваете документ на экране, достаточно навести указатель мыши на ссылку, и в строке состояния появится адрес перехода. В печатной версии пользователь не имеет ни малейшего представления, куда ведет ссылка.

Фирма AwesomeCo разрабатывает новую страницу со ссылками на бланки и регламентирующие документы. Один из членов комитета по переработке требует создания печатных копий страницы. Он хочет знать, куда ведет каждая ссылка на странице, чтобы определить, не нужно ли переместить ссылку в другое место. Необходимая функциональность легко реализуется средствами CSS; такое решение будет работать в IE8, Firefox, Safari и Chrome. Небольшой фрагмент кода JavaScript обеспечит его работоспособность в IE6 и 7.

Сама страница при этом представляет собой обычный список ссылок. Со временем она будет преобразована в шаблон.

css3_print_links/index.html

```
    <ul>
      <li>
        <a href="travel/index.html">Travel Authorization Form</
a>
      </li>
      <li>
        <a href="travel/expenses.html">Travel Reimbursement
        Form</a>
      </li>
      <li>
        <a href="travel/guidelines.html">Travel Guidelines</a>
      </li>
    </ul>

</body>
```

В печатной версии страницы совершенно неясно, на какой адрес ведет та или иная ссылка. Давайте исправим этот недостаток.

CSS

При добавлении таблицы стилей можно указать тип устройства вывода, для которого предназначено данное оформление. Чаще всего используется тип `screen` для вывода на экран, однако вы также можете использовать тип `print` для определения таблицы стилей, загружаемой только при выводе страницы на печать (или в режиме предварительного просмотра страницы).

`css3_print_links/index.html`

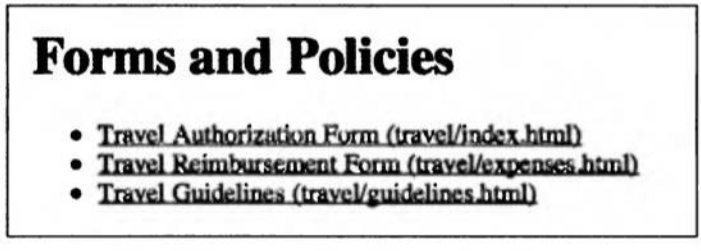
```
<link rel="stylesheet" href="print.css" type="text/css"
media="print">
```

Далее создается файл таблицы стилей *print.css* с простым правилом.

`css3_print_links/print.css`

```
a:after {
  content: " (" attr(href) ") ";
}
```

Это правило перебирает все ссылки на странице и добавляет к их тексту значение атрибута `href` в круглых скобках. При выводе на печать в современном браузере страница будет выглядеть так:



Если вы хотите посмотреть, как работает эта функция, но при этом не портить бумагу, — воспользуйтесь функцией предварительного просмотра страницы в браузере. В этом режиме также активизируется таблица стилей печати.

Решение работает во всех браузерах, кроме Internet Explorer 6 и 7. Эту проблему необходимо решить, не так ли?

Обходное решение

Internet Explorer поддерживает пару событий JavaScript, которые я хотел бы видеть в каждом браузере: `onbeforeprint` и `onafterprint`. При помощи этих событий можно изменить текст гиперссылки при активизации печати и вернуться к старым ссылкам при ее завершении. Пользователи и не заметят, что на странице что-то изменилось[1].

Создайте файл *print.js* и включите в него следующий код:

css3_print_links/print.js

```
$(function() {
  if (window.onbeforeprint !== undefined) {
    window.onbeforeprint = ShowLinks;
    window.onafterprint = HideLinks;
  }
});

function ShowLinks() {
  $("a").each(function() {
    $(this).data("linkText", $(this).text());
    $(this).append(" (" + $(this).attr("href") + ")");
  });
}

function HideLinks() {
  $("a").each(function() {
    $(this).text($(this).data("linkText"));
  });
}
```

Остается лишь связать файл со страницей. Обходное решение потребуется только для IE6 и 7, поэтому для включения будет использоваться условный комментарий. Так как работа решения зависит от jQuery, эту библиотеку тоже необходимо включить.

css3_print_links/index.html

```
<script
  charset="utf-8"
  src='http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/
  jquery.min.js'
```

[1] Метод хорошо описан в статье http://beckelman.net/post/2009/02/16/Use-jQuery-to-Show-a-Linke28099s-Address-After-its-Text-When-Printing-In-IE6-and-IE7.aspx

```
    type='text/javascript'>
  </script>
  <!--[if lte IE 7]>
  <script type="text/javascript" src="print.js"></script>
  <![endif]-->
</head>
<body>
  <h1>Forms and Policies</h1>

    <ul>
      <li>
        <a href="travel/index.html">Travel Authorization Form</a>
      </li>
      <li>
        <a href="travel/expenses.html">Travel Reimbursement
        Form</a>
      </li>
      <li>
        <a href="travel/guidelines.html">Travel Guidelines</a>
      </li>
    </ul>
```

После подключения кода JavaScript URL-адреса ссылок будут выводиться на печать во всех браузерах. На базе этой таблицы стилей печати можно построить другую, более полную. Кроме того, такое поведение может применяться только к некоторым ссылкам сайта (а не ко всем, как в нашем примере).

Рецепт 9. Создание многостолбцовых макетов

В издательском деле многостолбцовый вывод известен уже много лет; веб-дизайнеры смотрели на такие публикации с завистью. Узкие столбцы упрощают чтение контента, и с переходом на широкоэкранные дисплеи разработчики начали искать способы сохранения комфортной ширины столбцов. В конце концов, просматривать длинные строки на экране ничуть не удобнее, чем длинные строки в печатном бюллетене. За последнее десятилетие появился ряд довольно остроумных решений, но ни одно из них не сравнится по простоте и удобству с методом, представленным в спецификации CSS3.

Разбиение на столбцы

Фирма AwesomeCo публикует ежемесячный бюллетень для своих работников. Для работы с электронной почтой в фирме используется популярная веб-система. Бюллетени, рассылаемые по электронной почте, плохо выглядят и создают изрядные трудности с сопровождением. Было решено размещать бюллетень на интрасетевом сайте, чтобы в дальнейшем рассылать работникам ссылку для загрузки бюллетеня в браузере. Макет нового бюллетеня изображен на рис. 4.4.

Рис. 4.4. Одностолбцовый бюллетень плохо читается из-за слишком широких строк

Новый директор по связям с общественностью обладает опытом работы в издательской области. Он решил, что в электронном бюллетене, как и в печатном, информация должна выводиться в два столбца вместо одного.

Если вы когда-либо пытались разбить текст на столбцы с использованием `div` и `float`, то вы знаете, какой сложной порой оказывается эта задача. Первая серьезная проблема заключается в том, что вам приходится вручную выбирать точку разбивки текста. В издательских системах (таких, как InDesign) текстовые поля можно связать таким образом, что при заполнении одного поля текст автоматически «перетекает» в связанную область. В системах веб-разработки ничего похожего пока нет, но существует достаточно простой метод, который работает ничуть ни хуже. Мы можем взять элемент и разбить его содержимое на несколько столбцов одинаковой ширины.

Начнем с разметки бюллетеня. Она представляет собой весьма обычный HTML-код. Так как контент будет изменяться со временем, мы воспользуемся условным заполняющим текстом. Вас удивляет, что в разметке бюллетеня не используются новые элементы разметки (такие, как `section`)? Дело в том, что наше обходное решение несовместимо с этими элементами в Internet Explorer.

css3columns/condensed_newsletter.html

```
<body>
  <div id="header">
    <h1>AwesomeCo Newsletter</h1>
    <p>Volume 3, Issue 12</p>
  </div>
  <div id="newsletter">
    <div id="director_news">
      <div>
        <h2>News From The Director</h2>
      </div>
      <div>
        <p>
          Lorem ipsum dolor...
        </p>
        <div class="callout">
          <h4>Being Awesome</h4>
          <p>
            &amp;quot;Lorem ipsum dolor sit amet...&amp;quot;
          </p>
```

```
        </div>
        <p>
          Duis aute irure...
        </p>
      </div>
    </div>

    <div id="awesome_bits">
      <div>
        <h2>Quick Bits of Awesome</h2>
      </div>
      <div>
        <p>
          Lorem ipsum...
        </p>
        <p>
          Duis aute irure...
        </p>
      </div>
    </div>

    <div id="birthdays">
      <div>
        <h2>Birthdays</h2>
      </div>
      <div>
        <p>
          Lorem ipsum dolor...
        </p>
        <p>
          Duis aute irure...
        </p>
      </div>
    </div>

  </div>
  <div id="footer">
    <h6>Send newsworthy things to
      <a href="mailto:news@aweseomco.com">news@awesomeco.com</a>.
    </h6>
  </div>
</body>
```

Чтобы разбить этот контент на два столбца, достаточно включить в таблицу стилей следующий фрагмент:

css3columns/newsletter.html

```
#newsletter{
  -moz-column-count: 2;
  -webkit-column-count: 2;
  -moz-column-gap: 20px;
  -webkit-column-gap: 20px;
  -moz-column-rule: 1px solid #ddccb5;
  -webkit-column-rule: 1px solid #ddccb5;
}
```

ВОПРОС/ОТВЕТ

Можно ли задать разные ширины столбцов?

Нет, нельзя. Все столбцы должны иметь одинаковую ширину. Меня это тоже немного удивило, поэтому я лишний раз проверил спецификацию. На момент написания книги возможность определения столбцов разной ширины не поддерживалась.

Впрочем, если задуматься над тем, как традиционно используются столбцы, такое решение выглядит логично. Столбцы вовсе не предназначены для ухищрений с созданием боковых панелей на сайте — как, например, и таблицы. Столбцы предназначены для упрощения чтения длинных текстовых блоков, и столбцы равной ширины идеально подходят для этой цели.

Рис. 4.5. Новый двухстолбцовый бюллетень

Новая версия бюллетеня (рис. 4.5) выглядит намного лучше. При добавлении контента браузер автоматически определит, как выполнить равномерное разбиение. Также обратите внимание на то, что элементы со свойством `float` размещаются по границам содержащих их столбцов.

Обходное решение

Столбцы CSS3 не работают в Internet Explorer 8 и более ранних версий, поэтому нам понадобится обходное решение, основанное на использовании плагина jQuery Columnizer[1]. Columnizer обеспечивает равномерное разбиение контента на столбцы кодом следующего вида:

css3columns/newsletter.html

```
$("#newsletter").columnize({ columns: 2 });
```

Пользователям с отключенной поддержкой JavaScript придется довольствоваться одним столбцом текста. Впрочем, они хотя бы смогут прочитать линейно размеченный контент, так что их интересы не пострадают. Однако из кода JavaScript можно проверить, поддерживаются ли некоторые элементы браузером. Если прочитанное свойство CSS существует, мы получим пустую строку. Если возвращается `null`, это значит, что свойство недоступно.

css3columns/newsletter.html

```
<script
  charset="utf-8"
  src='http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/
  jquery.
  min.js'
  type='text/javascript'>
</script>
<script
  charset="utf-8"
  src="javascripts/autocolumn.js"
  type='text/javascript'>
</script>
<script type="text/javascript">
  function hasColumnSupport(){
    var element = document.documentElement;
```

[1] http://welcome.totheinter.net/columnizer-jquery-plugin/

```
    var style = element.style;
    if (style){
      return typeof style.columnCount == "string" ||
        typeof style.MozColumnCount == "string" ||
        typeof style.WebkitColumnCount == "string" ||
        typeof style.KhtmlColumnCount == "string";
    }
    return null;
  }

  $(function(){
    if(!hasColumnSupport()){
      $("#newsletter").columnize({ columns: 2 });
    }
  });
</script>
```

Код просто проверяет поддержку столбцов, и если она отсутствует, — применяет плагин.

Обновив страницу в Internet Explorer, вы увидите двухстолбцовый бюллетень. Возможно, он будет не идеальным (как на рис. 4.6) и вам придется применить CSS или JavaScript для исправления некоторых элементов; это занятие предоставляется читателю для самостоятельной работы. При необходимости используйте условные комментарии для конкретных версий Internet Explorer, как в рецепте 7 на с. 77.

Разбиение контента на столбцы упрощает чтение. С другой стороны, на очень длинных страницах пользователей может раздражать необходимость возвращаться к началу. Будьте осмотрительны при использовании этой возможности.

Рецепт 10. Построение мобильных интерфейсов

Возможность определения таблиц стилей для конкретных устройств вывода появилась уже давно, но до настоящего времени при определении таблицы стилей указывался только общий тип устройства (как в рецепте 8 на с. 87). Медиазапросы CSS3[1] позволяют изменять представление страницы в зависимости от размера экрана посетителя. Веб-разработчики годами использовали JavaScript для получения информации о размере экрана пользователя; теперь то же самое легко делается исключительно на уровне таблиц стилей. При помощи медиазапросов можно получить следующую информацию:

- Разрешение.
- Ориентация (книжная или альбомная).
- Ширина и высота устройства.
- Ширина и высота окна браузера.

Таким образом, медиазапросы позволяют легко создавать альтернативные таблицы стилей для мобильных пользователей.

Оказывается, администрация AwesomeCo недавно заменила свои коммуникаторы BlackBerry на новенькие iPhone. Директор по маркетингу хочет поскорее увидеть версию шаблона блога, построенную нами на с. 28, для iPhone. Задача решается очень просто.

В текущей версии блога используется двухстолбцовый макет с основной областью контента и боковой панелью. Чтобы информация лучше читалась на iPhone, проще всего удалить `float`-элементы. В этом случае боковая панель будет размещаться под основным контентом, а читателям не нужно будет прокручивать страницу по горизонтали.

ВОПРОС/ОТВЕТ

А как же тип устройства вывода Handheld?

Тип устройства вывода Handheld был предназначен специально для мобильных устройств. Однако такие устройства обычно пытаются показывать «настоящий Интернет», поэтому они игнорируют тип устройства вывода и используют таблицу стилей для типа screen.

[1] http://www.w3.org/TR/css3-mediaqueries/

Чтобы это решение заработало, добавьте в конец таблицы стилей блога следующий фрагмент кода:

css3mediaquery/style.css

```
@media only screen and (max-device-width: 480px) {
  body{
    width:460px;
  }

  section#sidebar, section#posts{
    float: none;
    width: 100%;
  }
}
```

Код в фигурных скобках медиазапроса можно рассматривать как отдельную таблицу стилей, которая активизируется при выполнении условий запроса. В данном примере она изменяет размеры тела страницы, а двухстолбцовый макет преобразуется в одностолбцовый.

Медиазапросы также можно задействовать при использовании внешних таблиц стилей, таким образом, таблица стилей для мобильных устройств может храниться в отдельном файле:

```
<link rel="stylesheet" type="text/css"
  href="CSS/mobile.css" media="only screen and (max-device-
  width: 480px)">
```

После внесения этого изменения блог будет нормально читаться на iPhone.

Описанным способом также можно создать таблицы стилей для других устройств — стендов, планшетов, экранов разного размера и т. д., чтобы ваш контент нормально читался на этих устройствах.

Обходное решение

Медиазапросы поддерживаются в Firefox, Chrome, Safari, Opera и Internet Explorer 9. Для загрузки дополнительных таблиц стилей в зависимости от пользовательского устройства приходится использовать обходные решения на базе JavaScript. Наш пример ориентирован на iPhone, и обходное решение в нем не требуется — контент будет читаться и без медиазапроса.

Но если вам захочется поэкспериментировать с медиазапросами в других браузерах, существует специальный плагин jQuery, реализующий поддержку базовых медиазапросов в других браузерах[1]. Учтите, что он работает только с внешними таблицами стилей и поддерживает только `min-width` и `max-width` в пикселах. Даже с учетом этих ограничений плагин очень удобен при создании разных интерфейсов для разных размеров окон.

Перспективы

Материал этой главы способствует совершенствованию пользовательского интерфейса, однако пользователи смогут работать с вашими веб-приложениями даже в том случае, если их браузеры не поддерживают новые возможности. Пользователь может прочитать данные в таблице, строки которой не окрашены в чередующиеся цвета; формы работают, даже если элементы интерфейса не украшены закруглением уголков; содержимое бюллетеня можно прочитать и в том случае, если оно выводится в один столбец. Хорошо, что все описанные эффекты могут быть реализованы на уровне представления, без применения JavaScript или решений на стороне сервера.

В настоящее время эти селекторы поддерживаются почти всеми браузерами, за исключением Internet Explorer. Вероятно, с течением времени IE тоже будет поддерживать их (особенно псевдоклассы). Из окончательной версии спецификации будут исключены специализированные префиксы вида `moz` и `webkit-`. Когда это произойдет, код обходного решения можно будет удалить.

---

[1] http://plugins.jquery.com/project/MediaQueries

5 Улучшение доступности

Многие новые элементы HTML5 предназначены для более точного описания контента. Они начинают играть более важную роль, когда ваш код интерпретируется другими программами. Например, некоторые пользователи используют специальные программы, называемые *экранными дикторами*, для преобразования графического содержимого экрана в произносимый текст. Экранный диктор интерпретирует код на экране и соответствующую разметку для идентификации ссылок, графики и других элементов. За последнее время в этой области были достигнуты выдающиеся результаты, но экранные дикторы всегда отстают от современных тенденций веб-разработки. Им трудно обнаружить активные области, в которых механизм опроса или запросы Ajax изменяют контент страниц. На более сложных страницах у пользователя возникают трудности с навигацией, потому что экранному диктору приходится зачитывать вслух описания слишком большого объема контента.

Спецификация WIA-ARIA[1] (Accessibility for Rich Internet Applications) определяет способы улучшения доступности веб-сайтов и особенно веб-приложений. Она чрезвычайно полезна при разработке приложений с элементами JavaScript и Ajax. Некоторые компоненты спецификации WIA-ARIA были включены в HTML5, другие существуют отдельно и могут дополнять спецификацию HTML5. Возможности WIA-ARIA уже используются многими экранными дикторами, включая JAWS, WindowEyes и даже встроенную функциональность Apple VoiceOver. WIA-ARIA также вводит дополнительную разметку, которая может использоваться вспомогательными устройствами для обнаруже-

[1] http://www.w3.org/WAI/intro/aria.php

ния обновляемых областей. В этой главе вы узнаете, как HTML5 упрощает работу пользователей вспомогательных устройств. Очень важно, что возможности, описанные в этой главе, не требуют дополнительных обходных решений, потому что многие экранные дикторы уже поддерживают их прямо сейчас.

К числу таких возможностей относятся[1]:

Атрибут `role` [`<div role="document">`]

Описание обязанностей элементов для экранных дикторов. [C3, F3.6, S4, IE8, O9.6]

`aria-live` [`<div aria-live="polite">`]

Идентификация автоматически обновляемых (вероятно, средствами Ajax) областей. [F3.6 (Windows), S4, IE8]

`aria-atomic` [`<div aria-live="polite" aria-atomic="true">`]

Признак чтения всего контента активной области или только изменившихся элементов. [F3.6 (Windows), S4, IE8]

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

Рецепт 11. Роли ARIA и упрощение навигации

Многие сайты используют распространенную структуру: заголовок, навигационная область, основной контент и завершитель. Большинство таких сайтов кодируется соответствующим образом, то есть линейно. К сожалению, это означает, что экранному диктору придется прочитать сайт своему пользователю в указанном порядке. Так как на многих сайтах заголовок и область навигации повторяются на каждой странице, пользователю придется прослушивать их при каждом посещении очередной страницы. Спецификация рекомендует решать эту проблему при помощи скрытой ссылки «пропуска навигации», которую экранный диктор будет зачитывать вслух; эта ссылка связывается с якорем где-то поблизости от основного контента. Однако такая функциональность не встраивается в страницы автоматически, и не все разработчики знают (или помнят), как она правильно реализуется.

Новый атрибут HTML5 `role` позволяет назначить «обязанность» каждому элементу страницы. Экранный диктор легко разбирает страницу и разбивает обязанности на категории для построения упрощенного представления страницы. Например, он может найти все роли `navigation` на странице и представить их пользователю для ускорения навигации по приложению.

Роли были взяты из спецификации WIA-ARIA[1] и интегрированы в спецификацию HTML5. Прямо сейчас можно использовать две конкретных разновидности ролей: *роли ориентиров* и *роли структуры документов*.

Роли ориентиров

Роли ориентиров идентифицируют «достопримечательности» вашего сайта (баннер, области поиска, область навигации и т. д.), чтобы упростить их обнаружение экранными дикторами.

Роль	Использование
banner	Область баннера на странице
search	Область поиска на странице
navigation	Элементы навигации на странице
main	Начало основного контента страницы

[1] http://www.w3.org/WAI/PF/aria/roles

Роль	Использование
contentinfo	Информация о контенте: авторское право, дата публикации и т. д.
complementary	Информация, дополняющая основной контент, но имеющая самостоятельный смысл
application	Область страницы, содержащая веб-приложение (в отличие от веб-документа)

Некоторые из ролей можно применить к шаблону блога AwesomeCo, над которым мы работали на с. 28. К общему заголовку применяется роль `banner`:

`html5_aria/blog/index.html`

```
<header id="page_header" role="banner">
  <h1>AwesomeCo Blog!</h1>
</header>
```

Идентификация роли осуществляется простым включением атрибута `role="banner"` в существующий тег.

Аналогичным образом определяется блок навигации.

`html5_aria/blog/index.html`

```
<nav role="navigation">
  <ul>
    <li><a href="/">Latest Posts</a></li>
    <li><a href="/archives">Archives</a></li>
    <li><a href="/contributors">Contributors</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
</nav>
```

В спецификации HTML5 указано, что некоторым элементам назначаются роли по умолчанию, которые не могут переопределяться. Элемент `nav` всегда имеет роль `navigation`, которую формально определять не нужно. Экранные дикторы еще не полностью поддерживают концепцию роли по умолчанию, но многие из них понимают роли ARIA.

Основная область и боковая панель определяются следующим образом:

`html5_aria/blog/index.html`

```
<section id="posts" role="main">
</section>
```

html5_aria/blog/index.html

```
<section id="sidebar" role="complementary">

  <nav>
    <h3>Archives</h3>
    <ul>
      <li><a href="2010/10">October 2010</a></li>
      <li><a href="2010/09">September 2010</a></li>
      <li><a href="2010/08">August 2010</a></li>
      <li><a href="2010/07">July 2010</a></li>
      <li><a href="2010/06">June 2010</a></li>
      <li><a href="2010/05">May 2010</a></li>
      <li><a href="2010/04">April 2010</a></li>
      <li><a href="2010/03">March 2010</a></li>
      <li><a href="2010/02">February 2010</a></li>
      <li><a href="2010/01">January 2010</a></li>
    </ul>
  </nav>
  </section> <!-- sidebar -->
```

Информация об авторском праве и публикации включается в область завершителя при помощи роли `contentinfo`.

html5_aria/blog/index.html

```
<footer id="page_footer" role="contentinfo">
  <p>&copy; 2010 AwesomeCo.</p>
</footer> <!-- footer -->
```

Если бы в нашем блоге была область поиска, ее также можно было бы определить с соответствующей ролью.

Итак, основные ориентиры страницы определены. Давайте сделаем следующий шаг и определим некоторые структурные элементы документа.

Роли структуры документа

Роли структуры документа упрощают идентификацию частей статического контента экранными дикторами для улучшения навигации.

Многие роли структуры документа (статьи, заголовки) неявно определяются тегами HTML. Однако роль `document` тегами не определяется, а эта роль чрезвычайно полезна, особенно в приложениях со смешением динамического и статического контента. Например, в почтовом веб-клиенте роль документа может быть связана с элементом, содержащим тело сообщения электронной почты. Это может быть полезно, потому

что экранные дикторы часто реализуют альтернативные методы навигации с использованием клавиатуры. Когда в экранном дикторе фокус принадлежит элементу приложения, он должен разрешить клавиши, управляющие работой веб-приложения. Однако при передаче фокуса статическому контенту привязки клавиш экранного диктора должны работать иначе.

ВОПРОС/ОТВЕТ

Зачем нужны роли ориентиров, если у нас уже есть такие элементы, как nav и header?

На первый взгляд кажется, что роли ориентиров избыточны, однако они обеспечивают необходимую гибкость для ситуаций, в которых не могут использоваться новые элементы.

При помощи роли search можно направить пользователей к области страницы, содержащей не только поле поиска, но и ссылку на карту сайта, раскрывающийся список «быстрых ссылок» и другие элементы, которые помогут пользователям быстро найти нужную информацию (в отличие от направления к одному полю поиска).

В спецификации также определяется много других ролей, не имеющих аналогов среди новых элементов и полей форм.

Роль	Использование
document	Область с контентом документа (в отличие от контента приложения)
article	Композиция, образующая независимую часть документа
definition	Определение термина или предмета
directory	Список ссылок на группу (например, оглавление). Используется для статического контента
heading	Заголовок раздела страницы
img	Раздел, содержащий элементы графического изображения. Это может быть как собственно графика, так и заголовки с текстом описания
list	Группа неинтерактивных элементов списка
listitem	Один неинтерактивный элемент списка
math	Математическое выражение
note	Контент, который является побочным или подчиненным по отношению к основному контенту ресурса
presentation	Контент, ориентированный на представление; может игнорироваться вспомогательными устройствами
row	Строка ячеек таблицы
rowheader	Ячейка с информацией заголовка для строки таблицы

В нашем блоге роль `document` логично добавить в элемент `body`.

`html5_aria/blog/index.html`

```
<body role="document">
```

Благодаря определению роли экранный диктор интерпретирует эту страницу как статический контент.

Обходное решение

Новейшие версии браузеров и экранных дикторов уже поддерживают роли, так что вы можете начать использовать их прямо сейчас. Если браузер не поддерживает роль, он просто проигнорирует ее; таким образом, определение ролей помогает только тем пользователям, которые могут их использовать.

Рецепт 12. Создание обновляемых областей с улучшенной доступностью

В наши дни технологии Ajax широко применяются в веб-приложениях. Один из стандартных приемов — визуальные эффекты, которые сообщают пользователю о неких изменениях на странице. Однако пользователь экранного диктора, естественно, этих эффектов не увидит.

В спецификации WIA-ARIA представлено неплохое альтернативное решение, которое в настоящее время работает в IE, Firefox и Safari с различными популярными экранными дикторами.

Директор по коммуникациям фирмы AwesomeCo хочет создать новую домашнюю страницу. На странице должны быть размещены ссылки на разделы услуг (`Service`), контактов (`Contacts`) и сведений о фирме (`About Us`). Он настаивает на том, что домашняя страница не должна прокручиваться, потому что пользователи «терпеть не могут прокрутку». Вам поручено реализовать прототип страницы с горизонтальным меню, которое изменяет основной контент страницы по щелчку. Это делается достаточно просто, а с атрибутом `aria-live` мы можем сделать то, что раньше было очень трудно осуществить, — создать реализацию такого интерфейса, хорошо сочетающуюся с экранными дикторами.

Примерный вид интерфейса изображен на рис. 5.1. Весь контент размещается на домашней странице; при поддержке JavaScript все части, кроме первой, скрываются. Мы создаем навигационные ссылки на разделы с использованием якорей страниц, а jQuery преобразует якорные ссылки в события, переключающие основной контент страницы. Пользователи с включенной поддержкой JavaScript увидят, что требует заказчик, а пользователи без JavaScript увидят весь контент страницы.

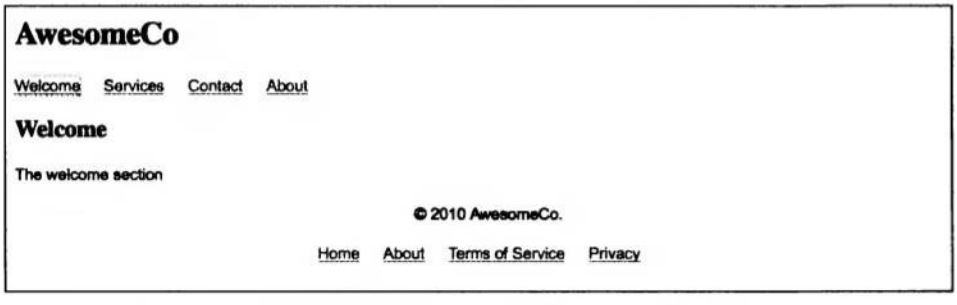


Рис. 5.1. Макет страницы с переключением контента средствами jQuery

Создание страницы

Начнем с создания базовой страницы HTML5. В страницу добавляется раздел приветствия, который отображается по умолчанию при посещении страницы пользователем. Код страницы с панелью навигации и ссылками переходов выглядит так:

html5_aria/homepage/index.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8" />
    <title>AwesomeCo</title>
    <link rel="stylesheet" href="style.css" type="text/css">
  </head>
  <body>
    <header id="header">
      <h1>AwesomeCo </h1>
      <nav>
        <ul>
          <li><a href="#welcome">Welcome</a></li>
          <li><a href="#services">Services</a></li>
          <li><a href="#contact">Contact</a></li>
          <li><a href="#about">About</a></li>
        </ul>
      </nav>
    </header>
    <section id="content"
             role="document" aria-live="assertive" aria-
             atomic="true">

      <section id="welcome">
        <header>
          <h2>Welcome</h2>
        </header>
        <p>The welcome section</p>
      </section>
    </section>
    <footer id="footer">
      <p>&copy; 2010 AwesomeCo.</p>
      <nav>
        <ul>
          <li><a href="http://awesomeco.com/">Home</a></li>
```

```
          <li><a href="about">About</a></li>
          <li><a href="terms.html">Terms of Service</a></li>
          <li><a href="privacy.html">Privacy</a></li>
        </ul>
      </nav>
    </footer>

</body>
</html>
```

Разделу приветствия присваивается идентификатор `welcome`, соответствующий якорю в области навигации. Другие разделы страницы объявляются аналогичным образом.

html5_aria/homepage/index.html

```
<section id="services">
  <header>
    <h2>Services</h2>
  </header>
  <p>The services section</p>
</section>

<section id="contact">
  <header>
    <h2>Contact</h2>
  </header>
  <p>The contact section</p>
</section>

<section id="about">
  <header>
    <h2>About</h2>
  </header>
  <p>The about section</p>
</section>
```

Четыре области контента заключаются в следующую разметку:

html5_aria/homepage/index.html

```
<section id="content"
         role="document" aria-live="assertive" aria-
         atomic="true">
```

Атрибуты в этой строке сообщают экранному диктору, что область страницы обновляется.

Методы обновления

Существует два метода оповещения пользователя об изменении страницы при использовании `aria-live`. Метод `polite` не прерывает рабочий процесс пользователя. Например, если экранный диктор пользователя зачитывает предложение, а другая область страницы обновляется в режиме `polite`, то экранный диктор дочитает предложение. С другой стороны, обновление в режиме `assertive` считается высокоприоритетным, поэтому экранный диктор прекратит чтение и перейдет к новому контенту. Очень важно правильно выбрать тип обновления при разработке сайта. Злоупотребление типом `assertive` может привести в замешательство пользователей. Используйте `assertive` только в случае крайней необходимости. В нашем случае этот выбор оправдан, потому что остальной контент скрывается.

Атомарное обновление

Второй параметр, `aria-atomic=true`, приказывает экранному диктору зачитать все содержимое измененной области. Если присвоить ему значение `false`, экранный диктор будет зачитывать только изменившиеся узлы. Мы заменяем весь контент, так что полное зачитывание в данном случае оправдано. Если бы мы заменяли только один элемент списка или присоединяли к таблице новую строку средствами Ajax, то было бы правильнее использовать значение `false`.

Скрываемые области

Чтобы скрыть области, мы напишем небольшой фрагмент кода JavaScript и присоединим его к странице. Создайте файл *application.js* и включите его вместе с библиотекой jQuery в страницу.

html5_aria/homepage/index.html

```
<script type="text/javascript"
  charset="utf-8"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.
min.js">
</script>

<script type="text/javascript"
  charset="utf-8"
  src="javascripts/application.js">
</script>
```

Файл *application.js* содержит следующий простой сценарий:

html5_aria/homepage/javascripts/application.js

```
 1   // Поддержка структурных элементов HTML5 для IE6, 7 и 8
-   document.createElement("header");
-   document.createElement("footer");
-   document.createElement("section");
 5   document.createElement("aside");
-   document.createElement("article");
-   document.createElement("nav");
-
-  $(function(){
10
-     $("#services, #about, #contact").hide().addClass("hidden");
-     $("#welcome").addClass("visible");
-
-     $("nav ul").click(function(event){
15
-       target = $(event.target);
-       if(target.is("a")){
-         event.preventDefault();
-         if ( $(target.attr("href")).hasClass("hidden") ){
20         $(".visible").removeClass("visible")
-             .addClass("hidden")
-             .hide();
-           $(target.attr("href"))
-             .removeClass("hidden")
25           .addClass("visible")
-             .show();
-         };
-       };
-
30   });
-
-  });
```

В строке 11 скрываются разделы `services`, `about` и `contact`. К ним также применяется класс `"hidden"`, а в следующей строке к разделу `"welcome"` применяется класс `"visible"`. Добавление классов позволяет легко определить включаемые и отключаемые разделы при переключении.

В строке 14 перехватываются все щелчки в области навигации, после чего в строке 17 мы определяем, на каком элементе был сделан щелчок.

Если пользователь щелкнул на ссылке, мы проверяем, скрыт ли соответствующий раздел. Атрибут `href` ссылки, на которой был сделан щелчок, позволяет легко найти соответствующий раздел при помощи селекторов jQuery (строка 19).

Если раздел скрыт, мы скрываем все остальное, а затем отображаем выбранный раздел. Вот и все, что требуется сделать, — экранные дикторы смогут обнаруживать изменения областей.

Обходное решение

Это решение, как и роли, может использоваться прямо сейчас новейшими версиями экранных дикторов. Оно просто реализовано, соответствует рекомендациям (ненавязчивое использование JavaScript) и работает для достаточно широкой аудитории. Дополнительные атрибуты игнорируются старыми браузерами и экранными дикторами, поэтому их включение в разметку ничем не грозит.

Перспективы

HTML5 и спецификация WIA-ARIA ведут к значительному улучшению доступности Web. Благодаря возможности определения изменяющихся областей страницы разработчики могут создавать приложения JavaScript с расширенной функциональностью, не беспокоясь о проблемах доступности.

II

Графика и звук

6 Рисование на «холсте»

Во второй части книги мы перейдем от структуры и интерфейсов к использованию HTML5 и CSS3 для графического вывода, работы с мультимедийными файлами и создания собственных элементов интерфейса. Начнем с создания графики с использованием нового элемента HTML5 `canvas`.

Чтобы включить графическое изображение в веб-приложение, традиционно веб-разработчику приходилось запускать свой любимый графический редактор, создавать графику и встраивать ее в страницу в теге `img`. Если ему была нужна анимация, приходилось использовать Flash. Элемент HTML5 `canvas` позволяет разработчику создавать графические изображения и анимации в браузере на программном уровне, средствами JavaScript. На созданном «холсте» можно рисовать простые или сложные фигуры и даже строить графики и диаграммы без использования серверных библиотек, Flash или специальных плагинов. Обе возможности будут продемонстрированы в этой главе[1].

`<canvas>` [`<canvas id="my_canvas" width="150" height="150">`]

Создание векторной графики из кода JavaScript. [C4, F3, IE9, S3.2, O10.1, IOS3.2, A2]

Начнем с совместного использования JavaScript с элементом `canvas` на примере построения логотипа AwesomeCo из простых геометрических фигур. Затем мы воспользуемся библиотекой построения графиков,

---

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

разработанной специально для работы с `canvas`, для создания гистограммы статистики браузеров. Также будут описаны некоторые проблемы с обходными решениями, с которыми мы столкнемся, потому что `canvas` скорее представляет собой программный интерфейс, нежели элемент.

Рецепт 13. Рисование логотипа

Элемент canvas является контейнером (как и элемент script). Он создает пустой холст, на котором можно рисовать. При создании холста указывается его ширина и высота.

html5canvasgraph/canvas_simple_drawing.html

```
<canvas id="my_canvas" width="150" height="150">
  Fallback content here
</canvas>
```

К сожалению, невозможно изменить ширину или высоту элемента canvas средствами CSS без искажения содержимого, поэтому размеры холста должны определяться на момент объявления.

Для рисования на холсте используется JavaScript. Даже если вы предоставляете обходной контент для браузеров, не поддерживающих элемент canvas, вам все равно необходимо предотвратить попытки работы с ним из кода JavaScript. Найдите элемент canvas по идентификатору и проверьте, поддерживает ли браузер его метод getContext.

html5canvasgraph/canvas_simple_drawing.html

```
var canvas = document.getElementById('my_canvas');
if (canvas.getContext){
  var context = canvas.getContext('2d');

}else{
  // Сделать что-то для вывода скрытого содержимого
  // или позволить браузеру вывести текст элемента <canvas>.
}
```

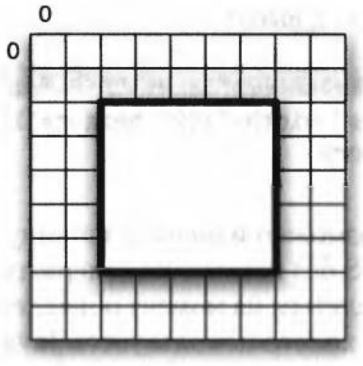
Если метод getContext возвращает ответ, мы получаем 2D-контекст холста для добавления объектов. Если ответа нет, необходимо найти способ отображения обходного контента.

Получив контекст холста, вы просто добавляете элементы в этот контекст. Например, чтобы добавить красный прямоугольник, следует установить цвет заливки, а потом создать фигуру.

html5canvasgraph/canvas_simple_drawing.html

```
context.fillStyle = "rgb(200,0,0)";
context.fillRect (10, 10, 100, 100);
```

По умолчанию начало координат 2D-контекста холста располагается в левом верхнем углу. При создании фигуры задаются координаты начальной точки *X* и *Y*, ширина и высота.



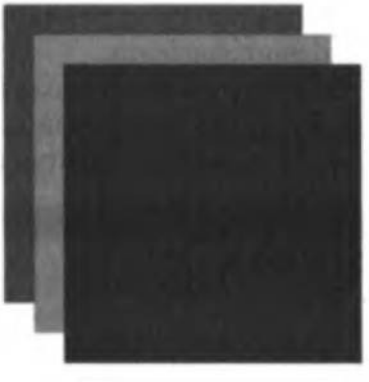
Каждая добавляемая фигура размещается на отдельном уровне; таким образом, можно создать три прямоугольника со смещением в 10 пикселов.

html5canvasgraph/canvas_simple_drawing.html

```
context.fillStyle = "rgb(200,0,0)";
context.fillRect (10, 10, 100, 100);
context.fillStyle = "rgb(0,200,0)";
context.fillRect (20, 20, 100, 100);

context.fillStyle = "rgb(0,0,200)";
context.fillRect (30, 30, 100, 100);
```

Созданные прямоугольники накладываются друг на друга.



Итак, вы примерно представляете основные принципы рисования на холсте, и мы можем перейти к рисованию логотипа AwesomeCo. Логотип, приведенный на рис. 6.1, выглядит достаточно простым.

Рис. 6.1. Логотип AwesomeCo

Рисование логотипа

Логотип состоит из строки текста, ломаной линии, квадрата и треугольника. Создайте новый документ HTML5, добавьте на страницу элемент `canvas`, а затем напишите функцию JavaScript для рисования логотипа. Функция проверяет возможность использования 2D-контекста.

html5canvasgraph/logo.html

```
  var drawLogo = function(){
    var canvas = document.getElementById('logo');
    var context = canvas.getContext('2d');
};
```

Перед вызовом метода сначала проверяется существование элемента `canvas`.

html5canvasgraph/logo.html

```
$(function(){
  var canvas = document.getElementById('logo');
  if (canvas.getContext){
    drawLogo();
  }
});
```

Так как на странице ищется элемент с идентификатором `logo`, для успешного поиска в документ необходимо добавить элемент `canvas` с соответствующим идентификатором.

html5canvasgraph/logo.html

```
<canvas id="logo" width="900" height="80">
  <h1>AwesomeCo</h1>
</canvas>
```

Следующим шагом станет вывод на холсте текста «AwesomeCo».

Добавление текста

Для вывода текста на холсте следует выбрать шрифт, размер шрифта и тип выравнивания, а затем вывести текст с указанием координат сетки. Вывод текста «AwesomeCo» происходит следующим образом:

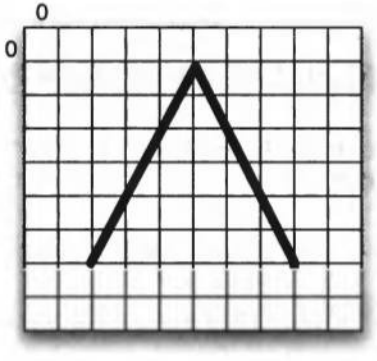
html5canvasgraph/logo.html

```
context.font = 'italic 40px sans-serif';
context.textBaseline = 'top';
context.fillText('AwesomeCo', 60, 0);
```

Перед применением текста к холсту мы определяем тип текста и задаем положение базовой линии (вертикальное выравнивание). Метод `fillText` заполняет текст цветом заливки, а вывод сдвигается на 60 пикселов вправо. Сдвиг освобождает место для треугольной ломаной, которая будет нарисована следующей.

Рисование линий

Рисование линий на холсте выполняется по принципу головоломки с соединением точек. Мы указываем начальную точку в координатах холста, а затем последовательно указываем точки для перемещения. В ходе перемещения по холсту точки соединяются отрезками.



Метод `beginPath()` начинает построение траектории, которое осуществляется последовательными вызовами `lineTo`:

html5canvasgraph/logo.html

```
context.lineWidth = 2;
context.beginPath();
```

```
context.moveTo(0, 40);
context.lineTo(30, 0);
context.lineTo(60, 40 );
context.lineTo(285, 40 );
context.stroke();
context.closePath();
```

После завершения перемещения по холсту вызов метода `stroke` рисует линию, а вызов `closePath` прекращает рисование.

Остается нарисовать рамку и комбинацию из треугольников внутри большого треугольника.

Перемещение начала координат

Нам нужно нарисовать маленький квадрат и треугольник внутри большого треугольника. При рисовании фигур и траекторий координаты *X* и *Y* могут задаваться относительно начала координат, находящегося в левом верхнем углу холста, однако начало координат также можно переместить в другую точку.

Рисуем меньший внутренний квадрат со смещением начала координат.

`html5canvasgraph/logo.html`

```
context.save();
context.translate(20,20);
context.fillRect(0,0,20,20);
```

Обратите внимание на вызов метода `save()` перед перемещением начала координат: он позволяет легко вернуться к предыдущему состоянию холста. Вызов `save()` создает «точку восстановления», а последовательные вызовы образуют своего рода стек. При каждом вызове `save()` создается новая позиция. Когда все операции будут выполнены, вызов `restore()` восстанавливает верхнюю позицию из стека.

Для рисования внутреннего треугольника будут использоваться траектории, но вместо `stroke` вызывается метод `fill` — он создает иллюзию того, что треугольник «выпадает» из квадрата.

`html5canvasgraph/logo.html`

```
context.fillStyle = '#fff';
context.strokeStyle = '#fff';

context.lineWidth = 2;
```

```
context.beginPath();
context.moveTo(0, 20);
context.lineTo(10, 0);
context.lineTo(20, 20 );
context.lineTo(0, 20 );
context.fill();
context.closePath();
context.restore();
```

Перед началом рисования выбирается белый цвет линий и заливки (`#fff`). Затем мы рисуем линии, а поскольку начало координат было смещено, координаты задаются относительно левого верхнего угла только что нарисованного квадрата.

Работа почти закончена; осталось добавить немного цвета.

Добавление цвета

На предыдущей странице был приведен пример назначения цвета линии и заливки при использовании графических инструментов. Чтобы окрасить весь логотип в красный цвет, достаточно включить следующий фрагмент перед началом рисования:

html5canvasgraph/logo.html

```
context.fillStyle = "#f00";
context.strokeStyle = "#f00";
```

Но это слишком просто. Интереснее создать градиенты и назначить их линиям и заливкам.

html5canvasgraph/logo_gradient.html

```
// context.fillStyle = "#f00";
// context.strokeStyle = "#f00";
var gradient = context.createLinearGradient(0, 0, 0, 40);
gradient.addColorStop(0, '#a00'); // красный
gradient.addColorStop(1, '#f00'); // красный
context.fillStyle = gradient;
context.strokeStyle = gradient;
```

Мы создаем объект градиента и задаем цветовые направляющие. В нашем примере переход осуществляется между двумя оттенками красного, но при желании можно создать и градиентную радугу[1].

---

[1] Пожалуйста, не делайте этого!

Обратите внимание: цвет выводимых объектов должен задаваться перед их рисованием.

В результате выполнения всех описанных действий логотип готов, а читатель лучше понимает принцип рисования простых фигур на холсте. Однако в Internet Explorer версий до 9 элемент `canvas` не поддерживался. Давайте исправим положение.

Обходное решение

Фирма Google выпустила библиотеку ExplorerCanvas[1], благодаря которой большая часть Canvas API становится доступной для пользователей Internet Explorer. Для этого достаточно включить библиотеку в страницу.

html5canvasgraph/logo_gradient.html

```
<!--[if lte IE 8]>
<script src="javascripts/excanvas.js"></script>
<![endif]-->
```

И все *должно* отлично заработать в Internet Explorer, но пока еще не работает. На момент написания книги самая стабильная версия вообще не поддерживала добавление текста, а версия из репозитория Subversion[2] использовала неправильные шрифты. Кроме того, библиотека пока не поддерживает градиенты для рисования линий.

Итак, вместо этого приходится применять другие решения (например, размещать в элементе `canvas` графику логотипа в формате PNG) или просто не использовать `canvas`. Впрочем, наш пример всего лишь демонстрирует возможности графического вывода, так что если его не удастся использовать в кросс-платформенной реальной системе, ничего страшного не произойдет.

---

[1] http://code.google.com/p/explorercanvas/

[2] http://explorercanvas.googlecode.com/svn/trunk/excanvas.js

Рецепт 14. Построение диаграмм средствами RGraph

Фирма AwesomeCo вкладывает много труда в свой сайт, и высшее руководство хочет видеть график веб-статистики. Программисты, занимающиеся исполнительной подсистемой, могут получить данные в реальном времени. Однако сначала они хотят узнать, как вы собираетесь строить диаграмму в браузере, поэтому вам были предоставлены тестовые данные. Наша цель — по имеющимся тестовым данным построить диаграмму наподобие изображенной на рис. 6.2.

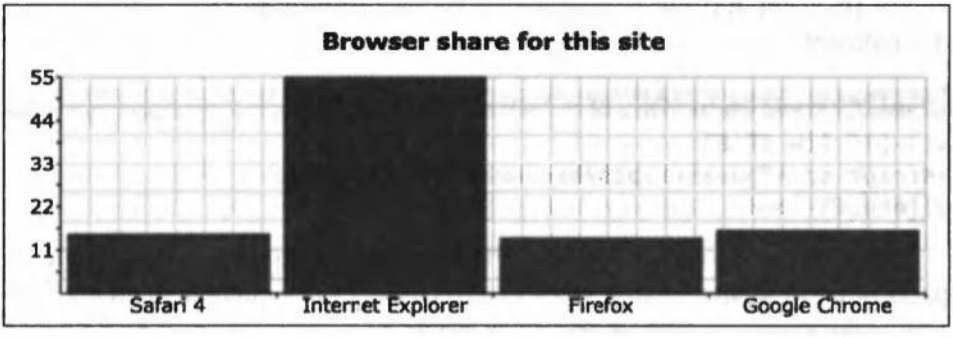


Рис. 6.2. Построение диаграммы на стороне клиента с использованием элемента canvas

Существует много разных средств для построения диаграмм и графиков на веб-страницах. Разработчики часто используют Flash, но у такого решения есть недостаток: оно не работает на некоторых мобильных устройствах (например, iPad и iPhone). Некоторые решения на стороне сервера работают хорошо, но создают чрезмерную нагрузку на процессор при работе с данными в реальном времени. Хорошим выходом из положения может стать стандартизированное решение клиентской стороны (такое, как `canvas`), но вы должны обеспечить его работоспособность в старых браузерах. Вы уже умеете рисовать прямоугольники, но для рисования сложных фигур потребуется существенно больший объем кода JavaScript. Для упрощения работы стоит воспользоваться библиотекой построения диаграмм.

Тот факт, что стандарт HTML5 еще не имеет повсеместной поддержки, не остановил разработчиков библиотеки RGraph[1]. С RGraph задача

[1] http://www.rgraph.net/

рисования диаграмм на холстах HTML5 становится до смешного простой. Впрочем, это решение базируется исключительно на JavaScript, так что оно не будет работать в пользовательских агентах с отключенной поддержкой JavaScript, как, впрочем, и поддержка canvas. Ниже приведен код построения очень простой гистограммы.

Download html5canvasgraph/rgraph_bar_example.html

```
<canvas width="500" height="250" id="test">[no canvas support]</
canvas>

<script type="text/javascript" charset="utf-8">
  var bar = new RGraph.Bar('test', [50,25,15,10]);
  bar.Set('chart.gutter', 50);
  bar.Set('chart.colors', ['red']);
  bar.Set('chart.title', "A bar graph of my favorite pies");
  bar.Set('chart.labels', ["Banana Creme", "Pumpkin", "Apple",
  "Cherry"]);
  bar.Draw();
</script>
```

От вас потребуется лишь создать пару массивов JavaScript, а библиотека построит диаграмму на холсте сама.

Описание данных в HTML

Статистику использования браузеров, по которой строится диаграмма, можно жестко закодировать в коде JavaScript, но тогда данные будут доступны только пользователям с включенной поддержкой JavaScript. Вместо этого мы разместим данные прямо на странице в текстовом виде. Позднее данные можно будет прочитать из JavaScript и передать их библиотеке построения диаграмм.

html5canvasgraph/canvas_graph.html

```
<div id="graph_data">
  <h1>Browser share for this site</h1>
  <ul>
    <li>
      <p data-name="Safari 4" data-percent="15">
        Safari 4 - 15%
      </p>
    </li>
    <li>
      <p data-name="Internet Explorer" data-percent="55">
```

```
        Internet Explorer - 55%
      </p>
    </li>
    <li>
      <p data-name="Firefox" data-percent="14">
        Firefox - 14%
      </p>
    </li>
    <li>
      <p data-name="Google Chrome" data-percent="16">
        Google Chrome - 16%
      </p>
    </li>
  </ul>
</div>
```

Мы используем атрибуты данных HTML5 для хранения имен браузеров и доли их использования в процентах. Хотя информация присутствует в текстовом виде, работать с атрибутами данных на программном уровне намного удобнее, потому что не приходится заниматься разбором строк.

Если вы откроете страницу в своем браузере или просто посмотрите на рис. 6.3, то увидите, что данные выводятся и читаются даже без диаграммы. Этот обходной контент будет использоваться мобильными устройствами и пользователями, чьи браузеры не поддерживают элемент `canvas` или JavaScript.

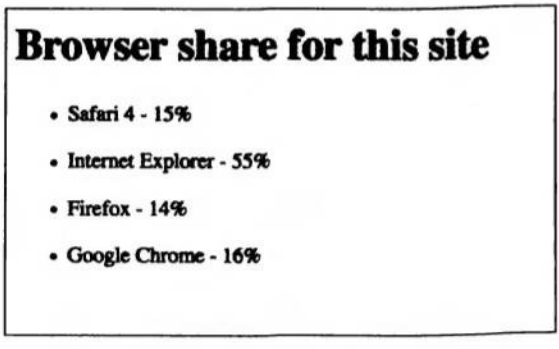


Рис. 6.3. Наша диаграмма в виде HTML

Давайте преобразуем эту разметку в диаграмму.

Преобразование кода HTML в гистограмму

Так как мы собираемся строить гистограмму, нам понадобится как основная библиотека RGraph, так и библиотека RGraph Bar graph. Также для извлечения данных из документа будет использоваться библиотека jQuery. Необходимые библиотеки загружаются в заголовочном разделе страницы HTML.

html5canvasgraph/canvas_graph.html

```
<script type="text/javascript"
  charset="utf-8"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.
min.js">
</script>
<script src="javascripts/RGraph.common.js" ></script>
<script src="javascripts/RGraph.bar.js" ></script>
```

Чтобы построить диаграмму, необходимо извлечь из HTML-документа название диаграммы, надписи и данные и передать их библиотеке RGraph. И метки, и данные передаются RGraph в массивах. Для быстрого построения обоих массивов можно воспользоваться jQuery.

html5canvasgraph/canvas_graph.html

```
 1  function canvasGraph(){
-     var title = $('#graph_data h1').text();
-
-     var labels = $("#graph_data>ul>li>p[data-name]").
                  map(function(){
 5       return $(this).attr("data-name");
-     });
-
-     var percents = $("#graph_data>ul>li>p[data-percent]").
                    map(function(){
-       return parseInt($(this).attr("data-percent"));
10    });
-
-     var bar = new RGraph.Bar('browsers', percents);
-     bar.Set('chart.gutter', 50);
-     bar.Set('chart.colors', ['red']);
15    bar.Set('chart.title', title);
-     bar.Set('chart.labels', labels);
-     bar.Draw();
-
-  }
```

В строке 2 мы читаем текст заголовка диаграммы. Далее, в строке 4 выбираются все элементы с атрибутом `data-name`. Для преобразования значений этих элементов в массив используется функция jQuery `map`.

Аналогичная логика используется в строке 8 для заполнения массива процентов.

После того как все данные будут собраны, RGraph легко строит диаграмму.

Отображение альтернативного контента

В рецепте 14 я бы мог разместить диаграмму между начальным и конечным тегами `canvas`. В этом случае элементы были бы скрыты в браузерах, поддерживающих `canvas`, и отображались бы в браузерах, в которых такая поддержка отсутствует. Однако если браузер поддерживает `canvas`, но пользователь отключил JavaScript, контент оставался бы скрытым.

JQUERY CSS И CSS

В этой главе мы использовали jQuery для применения стилей к элементам при создании. Большую часть стилевой информации (например, цвета меток и цвета столбцов) следует вынести в отдельную таблицу стилей, особенно если вы хотите иметь возможность менять стили независимо от сценария. Для прототипа сгодится и такое решение, но в реальном коде всегда следует разделять представление, поведение и контент.

Соответственно мы оставляем данные за пределами элемента `canvas`, а затем скрываем их средствами jQuery после проверки существования `canvas`.

`html5canvasgraph/canvas_graph.html`

```
var canvas = document.getElementById('browsers');
if (canvas.getContext){
  $('#graph_data').hide();
  canvasGraph();
}
```

Диаграмма готова — ее увидят все, кроме пользователей с браузерами, не поддерживающими элемент `canvas`.

Обходное решение

При построении решения уже упоминались некоторые обходные пути для улучшения доступности и отсутствия поддержки JavaScript. Однако

мы можем построить альтернативную диаграмму для пользователей, браузеры которых не поддерживают canvas, но могут использовать JavaScript.

Существует множество библиотек для построения диаграмм, но все они используют разные способы передачи данных. Гистограммы состоят из обычных прямоугольников с определенными высотами, а страница содержит все данные, по которым можно построить диаграмму вручную.

html5canvasgraph/canvas_graph.html

```
function divGraph(barColor, textColor, width, spacer,
                  lblHeight){
-    $('#graph_data ul').hide();
-    var container = $("#graph_data");
-
     container.css( {
-      "display" : "block",
-      "position" : "relative",
-      "height": "300px"}
-    );

-    $("#graph_data>ul>li>p").each(function(i){
-
-      var bar = $("<div>" + $(this).attr("data-percent") +
                  "%</ div>");
-      var label = $("<div>" + $(this).attr("data-name") +
                    "</ div>");

-      var commonCSS = {
-                    "width": width + "px",
-                    "position" : "absolute",
-                    "left" : i * (width + spacer) + "px"};

-      var barCSS = {
-                "background-color" : barColor,
-                "color" : textColor,
-                "bottom" : lblHeight + "px",
                "height" : $(this).attr("data-percent") + "%"
-      };
-      var labelCSS = {"bottom" : "0", "text-align" : "center"};
-
-      bar.css( $.extend(barCSS, commonCSS) );
      label.css( $.extend(labelCSS,commonCSS) );
-
```

```
-       container.append(bar);
-       container.append(label);
-     });
35
-   }
```

В строке 2 неупорядоченный список скрывается. Далее мы находим элемент, содержащий данные диаграммы, и применяем базовые стили CSS. В строке 7 выбирается режим позиционирования `relative`, чтобы столбцы и метки размещались в системе координат контейнера.

Затем мы перебираем абзацы маркированного списка (строка 11) и создаем столбцы гистограммы. Каждая итерация создает два элемента `div`, для самого столбца и для метки, которая располагается под ним. Таким образом, при помощи несложных математических вычислений и кода jQuery мы можем воссоздать гистограмму. Возможно, она не совпадает с оригиналом *полностью*, но достаточно близка к нему для демонстрации возможности.

Остается лишь связать альтернативное решение с логикой обнаружения `canvas`.

html5canvasgraph/canvas_graph.html

```
var canvas = document.getElementById('browsers');
if (canvas.getContext){
  $('#graph_data').hide();
  canvasGraph();
}
else{
  divGraph("#f00", "#fff", 140, 10, 20);
}
```

Альтернативная версия гистограммы изображена на рис. 6.4. Используя комбинацию JavaScript, HTML и CSS, мы реализовали построение диаграммы на стороне клиента и вывод статистической информации об использовании браузеров для любой платформы, для которой это может понадобиться. У использования `canvas` есть одно дополнительное преимущество: оно заставляет нас думать об обходном решении с самого начала (вместо попыток «прицепить» что-нибудь позднее). Безусловно, это полезно с точки зрения доступности.

Описанный метод диаграммного представления данных является одним из самых универсальных и доступных. Он позволяет легко создать визуальное представление вместе с текстовой альтернативой, чтобы все

пользователи могли работать с важными данными, отображаемыми на вашей странице.

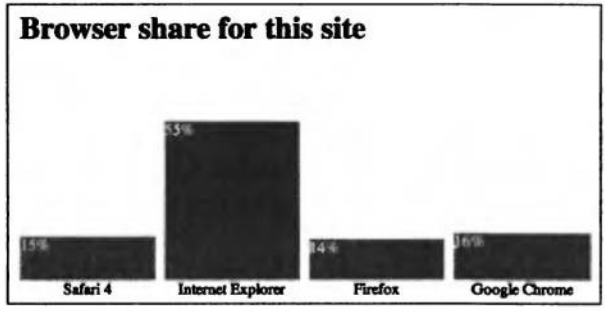


Рис. 6.4. Наша гистограмма теперь отображается в Internet Explorer

ВОПРОС/ОТВЕТ

А почему мы не воспользовались ExplorerCanvas?

Библиотеки ExplorerCanvas (см. рецепт 13) и RGraph хорошо сочетаются друг с другом. В дистрибутив RGraph даже включена версия ExplorerCanvas. Тем не менее эта комбинация работает только в Internet Explorer 8. В Internet Explorer 7 и более ранних версиях приходится использовать альтернативное решение вроде рассмотренного нами. Я рекомендую следить за развитием ExplorerCanvas, потому что библиотека активно развивается. Вы даже можете попробовать поэкспериментировать с ней, чтобы заставить работать в нужной для вас конфигурации.

Перспективы

Теперь вы примерно представляете, как работает элемент `canvas`, и можете найти новые возможности его использования. Например, на базе `canvas` можно создать игру, интерфейс для воспроизведения мультимедийных файлов или графическую галерею с расширенными возможностями. Все, что нужно для рисования — JavaScript и немного воображения.

В настоящее время решения на базе Flash предпочтительнее `canvas`, потому что они имеют более широкую поддержку. Тем не менее HTML5 постепенно набирает популярность, и элемент `canvas` становится доступным для более широкой аудитории; все больше разработчиков используют его для реализации простой 2D-графики в браузере. Элемент `canvas` не требует дополнительных плагинов и расходует меньше

ресурсов процессора, чем Flash (особенно в Linux и OS X). Наконец, 2D-графика на базе canvas может использоваться в средах, в которых поддержка Flash недоступна. По мере того как поддержка canvas реализуется на все большем количестве платформ, можно ожидать улучшения скорости и функциональности операций с canvas. Можно ожидать, что в будущем появятся новые инструменты разработчика и библиотеки с новыми интересными возможностями.

Однако возможности canvas не ограничиваются 2D-графикой. Со временем в спецификации canvas появится поддержка 3D-графики, а разработчики браузеров реализуют поддержку аппаратного ускорения. Веб-разработчики смогут создавать оригинальные пользовательские интерфейсы и увлекательные игры и для этого им не понадобится ничего, кроме JavaScript.

7

Внедрение видео и аудио

Аудио- и видеоматериалы стали важной частью современного Интернета. Подкасты, аудиокомментарии и даже видеоинструкции появились повсеместно, и до настоящего времени для их воспроизведения требовались специальные браузерные плагины. В HTML5 появились новые механизмы встраивания аудио- и видеофайлов в страницу. В этой главе будут рассмотрены некоторые методы, которые позволяют не только внедрять аудио- и видеоконтент, но и обеспечивают его доступность для пользователей старых браузеров.

В этой главе рассматриваются следующие два элемента[1]:

`<audio>` [`<audio src="drums.mp3"></audio>`]

Воспроизведение аудио встроенными средствами браузера. [C4, F3.6, IE9, S3.2, O10.1, IOS3, A2]

`<video>` [`<video src="tutorial.m4v"></video>`]

Воспроизведение видео встроенными средствами браузера. [C4, F3.6, IE9, S3.2, O10.5, IOS3, A2]

Но сначала необходимо припомнить историю работы с аудио и видео в Web. В конце концов, чтобы понять, куда мы идем, нужно знать, откуда мы вышли.

---

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

7.1. Немного истории

Попытки использования аудио и видео в веб-страницах начались довольно давно. Сначала разработчики внедряли MIDI-файлы в свои домашние страницы, и использовали тег `embed` для ссылок на них.

```
<embed src="awesome.mp3" autostart="true"
  loop="true" controller="true"></embed>
```

Тег `embed` так и не стал стандартным, поэтому вместо него стал использоваться тег `object`, принятый в качестве стандарта W3C. Для старых браузеров, не поддерживающих тег `object`, часто используется конструкция с вложением тега `embed` в `object`:

```
<object>
<param name="src" value="simpsons.mp3">
<param name="autoplay" value="false">
<param name="controller" value="true">
<embed src="awesome.mp3" autostart="false"
loop="false" controller="true"></embed>
</object>
```

Однако не каждый браузер мог осуществлять потоковое воспроизведение контента, и не каждый сервер был правильно настроен для работы с ним. Ситуация еще сильнее усложнилась с ростом популярности видео в Web. Работа с аудио- и видеоконтентом в Web прошла много итераций, от RealPlayer до Windows Media и QuickTime. Каждая компания имела свою стратегию работы с видео; порой казалось, что в Интернете не найти два сайта с одинаковыми методами и форматами кодирования видео.

Фирма Macromedia (ныне Adobe) довольно быстро осознала, что ее Flash Player может стать идеальным кросс-платформенным средством воспроизведения аудио- и видеоконтента. Технология Flash поддерживается примерно 97 % браузеров. Когда производители контента поняли, что контент после однократного кодирования можно воспроизводить где угодно, тысячи сайтов перешли на потоковые технологии Flash для работы с аудио и видео.

Затем в 2007 году фирма Apple выпустила iPhone и iPod Touch. Было решено, что Apple не будет поддерживать Flash на этих устройствах. Поставщики контента отреагировали на эту новость созданием видеопотоков, нормально воспроизводимых в браузере Mobile Safari. Эти видеопотоки, использующие кодек h>264, также могли воспроизводиться обычным проигрывателем Flash Player, что позволило поставщикам

контента, как и прежде, обеспечивать воспроизведение на разных платформах при однократном кодировании.

Создатели спецификации HTML5 полагают, что браузер должен иметь встроенную поддержку аудио и видео, вместо использования плагинов, требующих большого объема шаблонного кода HTML. Именно здесь проявляется основная стратегия работы с аудио и видео в HTML5: аудио и видео стали полноправными видами веб-контента.

ВОПРОС/ОТВЕТ

Flash уже работает почти во всех браузерах, так почему не использовать эту технологию?

Главная причина — отсутствие ограничений, устанавливаемых фирмой-разработчиком, на работу с контентом после его внедрения в страницу. Для операций с элементом можно использовать CSS и JavaScript, и вам не нужно возиться с передачей параметров Flash-роликам. Вдобавок ситуация будет улучшаться по мере «взросления» стандарта.

7.2. Контейнеры и кодеки

При обсуждении работы с видео в Web часто используются термины «контейнер» и «кодек». Контейнер можно сравнить с конвертом, содержащим аудио- и видеопотоки, а также иногда дополнительные метаданные (например, субтитры). Эти аудио- и видеопотоки должны быть как-то закодированы; эта задача решается при помощи кодеков. Существуют сотни разных способов кодирования аудио и видео, но в контексте видео HTML5 важны лишь несколько из них.

Видеокодеки

При просмотре видео проигрыватель должен декодировать его. К сожалению, может оказаться, что используемый проигрыватель не способен декодировать тот видеопоток, который вы хотите посмотреть. Некоторые проигрыватели используют программное декодирование видеопотока, которое может выполняться более медленно или сильнее расходовать ресурсы процессора. Другие проигрыватели используют аппаратное декодирование, а следовательно, их возможности воспроизведения более ограничены. Если вы хотите поскорее приступить к использованию тега HTML5 `video`, вам необходимо знать о трех видеоформатах: H.264, Theora и VP8.

Кодеки и их поддержка браузерами

H.264
[IE9, S4, C3, IOS]
Theora
[F3.5, C4, O10]
VP8
[IE9 (если кодек установлен), F4, C5, O10.7]

H.264

H.264 — высококачественный кодек, созданный группой MPEG и стандартизированный в 2003 году. Для обеспечения поддержки устройств с минимальными возможностями (например, сотовых телефонов) параллельно с поддержкой устройств высокого разрешения спецификация H.264 делится на профили. Они обладают некоторыми общими возможностями, но профили более высокого уровня предоставляют дополнительные возможности для улучшения качества. Например, iPhone и Flash Player могут воспроизводить видеопоток, закодированный по стандарту H.264, но iPhone поддерживает только низкокачественный «базовый» профиль, тогда как Flash Player поддерживает потоки более высокого качества. Видеопоток можно закодировать один раз с внедрением нескольких профилей — в этом случае он будет хорошо выглядеть на разных платформах.

H.264 является фактическим стандартом благодаря поддержке фирм Microsoft и Apple, которые являются обладателями лицензий. Кроме того, видеосервис Google YouTube преобразовал свои видеоролики с использованием кодека H.264, чтобы они могли воспроизводиться на iPhone; кроме того, H.264 поддерживается программой Flash Player фирмы Adobe. Тем не менее технология не является открытой. Кодек запатентован, а его использование определяется условиями лицензирования. Производители контента должны платить лицензионные отчисления за кодирование видео с использованием кодека H.264, однако эти отчисления не распространяются на контент, бесплатно предоставляемый конечному пользователю[1].

Сторонники свободного распространения ПО беспокоятся о том, что со временем правообладатели начнут требовать высоких отчислений от производителей контента. Это привело к созданию и распространению альтернативных кодеков.

---

[1] http://www.reelseo.com/mpeg-la-announces-avc-h264-free-license-lifetime/

Theora

Theora — бесплатный кодек, разработанный Xiph.Org Foundation. Хотя производители контента могут использовать Theora для создания видео качества, сопоставимого с H.264, производители устройств не торопятся с его поддержкой. Firefox, Chrome и Opera могут воспроизводить видео в формате Theora на любой платформе без дополнительного программного обеспечения, но Internet Explorer, Safari и устройства iOS такой поддержки не предоставляют. Apple и Microsoft опасаются «скрытых патентов» — этим термином обозначается намеренная задержка публикации и оформления патента для сохранения скрытности, пока другие реализуют технологию. Когда приходит подходящий момент, правообладатель патента «заявляет о себе» и начинает требовать лицензионные отчисления с ничего не подозревающего рынка.

VP8

VP8 фирмы Google — полностью открытый бесплатный кодек, по качеству сходный с H.264. Он поддерживается в Mozilla, Google Chrome и Opera, а Microsoft обещает обеспечить поддержку VP8 в Internet Explorer 9 при условии, что кодек уже установлен у пользователя. Кроме того, кодек поддерживается в Adobe Flash Player, что делает его интересной альтернативой. С другой стороны, кодек не поддерживается в Safari и устройствах iOS; таким образом, несмотря на бесплатность кодека, производителям видеоконтента для iPhone или iPad все равно приходится использовать кодек H.264.

Аудиокодеки

Ситуация с конкуренцией стандартов видеоконтента дополнительно усложняется наличием конкурирующих аудиостандартов.

Кодеки и их поддержка браузерами

AAC
  [S4, C3, IOS]
MP3
  [IE9, S4, C3, IOS]
Vorbis (OGG)
  [F3, C4, O10]

AAC (Advanced Audio Coding)

Фирма Apple использует этот аудиоформат в своем магазине iTunes Store. Кодек обеспечивает более высокое качество, чем MP3, при сходном размере файла, а также поддерживает несколько аудиопрофилей по аналогии с H.264. Кроме того, как и в случае с H.264, использование кодека требует лицензионных отчислений.

Файлы AAC воспроизводятся всеми продуктами Apple, а также Adobe Flash Player и проигрывателем с открытым кодом VLC.

Vorbis (OGG)

Бесплатный формат с открытым кодом, не требующий лицензионных отчислений; поддерживается в Firefox, Opera и Chrome. Может использоваться в сочетании с видеокодеками Theora и VP8. Файлы Vorbis обеспечивают хорошее качество звука, но недостаточно широко поддерживаются аппаратными проигрывателями.

MP3

Формат MP3, несмотря на свою исключительную популярность, не поддерживается в Firefox и Opera из-за патентных осложнений. С другой стороны, он поддерживается в Safari и Google Chrome.

Для воспроизведения и распространения видеоконтента видео- и аудиопотоки упаковываются в один контейнер. Итак, что же такое «видеоконтейнер»?

Контейнеры и кодеки: совместная работа

Контейнер представляет собой файл метаданных, который описывает и объединяет аудио- и видеофайлы. Контейнер не содержит информации о том, каким образом закодирована содержащаяся в нем информация. По сути, контейнер является «оберткой» для аудио- и видеопотоков. В общем случае контейнер может содержать произвольную комбинацию закодированных потоков, но при работе с видео в Web обычно используются следующие комбинации:

- Контейнер OGG с видео Theora и аудио Vorbis — работает в Firefox, Chrome и Opera.
- Контейнер MP4 с видео H.264 и аудио AAC — работает в Safari и Chrome. Также воспроизводится в Adobe Flash Player, на устройствах iPhone, iPod и iPad.

- Контейнер WebM с видео VP8 и аудио Vorbis — работает в Firefox, Chrome, Opera и Adobe Flash Player.

Google и Mozilla в своем развитии ориентируются на VP8 и WebM, так что Theora со временем можно будет исключить из рассмотрения. Однако текущее состояние дел таково, что мы все равно сталкиваемся с необходимостью двукратного кодирования видео — для пользователей Apple (которые занимают относительно малую нишу в секторе настольных компьютеров, но являются владельцами значительной части мобильных устройств в США) и для пользователей Firefox и Opera, так как оба этих браузера отказываются воспроизводить H.264[1].

Вводная часть получилась довольно длинной, но теперь вы понимаете историю вопроса и ограничения разных браузеров. Наше изучение непосредственной поддержки аудио- и видеоконтента начнется с аудио.

[1] http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2009-June/020620.html

Рецепт 15. Работа с аудио

Фирма AwesomeCo разрабатывает сайт с бесплатными аудиосемплами. Заказчик хочет видеть макет условной страницы с одной коллекцией семплов. На готовой странице должен размещаться список аудиосемплов, каждый из которых может быть легко прослушан посетителем. Беспокоиться о поиске семплов для проекта не нужно — звукорежиссер клиента уже предоставил необходимые файлы в форматах MP3 и OGG. Небольшой вводный курс самостоятельного кодирования аудиофайлов приведен в приложении B.

Построение списка

Звукорежиссер предоставил четыре семпла: `drums`, `organ`, `bass` и `guitar`. Мы должны описать каждый из этих семплов в разметке HTML. Разметка для семпла `drums` выглядит так:

html5_audio/audio.html

```
<article class="sample">
  <header><h2>Drums</h2></header>
  <audio id="drums" controls>
    <source src="sounds/ogg/drums.ogg" type="audio/ogg">
    <source src="sounds/mp3/drums.mp3" type="audio/mpeg">
    <a href="sounds/mp3/drums.mp3">Download drums.mp3</a>
  </audio>
</article>
```

Сначала мы определяем элемент `audio` и указываем, что на странице должны отображаться кнопки управления воспроизведением. Далее для элемента определяются несколько файлов-источников: сначала мы определяем MP3- и OGG-версии семпла, а затем выводим ссылку для загрузки файла MP3 на случай, если браузер не поддерживает элемент `audio`.

Этот базовый код работает в Chrome, Safari и Firefox. Давайте включим его в шаблон HTML5 с тремя другими звуковыми семплами.

html5_audio/audio.html

```
    <article class="sample">
      <header><h2>Drums</h2></header>
      <audio id="drums" controls>
        <source src="sounds/ogg/drums.ogg" type="audio/ogg">
```

```
      <source src="sounds/mp3/drums.mp3" type="audio/mpeg">
      <a href="sounds/mp3/drums.mp3">Download drums.mp3</a>
    </audio>
  </article>

  <article class="sample">
    <header><h2>Guitar</h2></header>
    <audio id="guitar" controls>
      <source src="sounds/ogg/guitar.ogg" type="audio/ogg">
      <source src="sounds/mp3/guitar.mp3" type="audio/mpeg">
      <a href="sounds/mp3/guitar.mp3">Download guitar.mp3</a>
    </audio>
  </article>

  <article class="sample">
    <header><h2>Organ</h2></header>
    <audio id="organ" controls>
      <source src="sounds/ogg/organ.ogg" type="audio/ogg">
      <source src="sounds/mp3/organ.mp3" type="audio/mpeg">
      <a href="sounds/mp3/organ.mp3">Download organ.mp3</a>
    </audio>
  </article>

  <article class="sample">
    <header><h2>Bass</h2></header>
    <audio id="bass" controls>
      <source src="sounds/ogg/bass.ogg" type="audio/ogg">
      <source src="sounds/mp3/bass.mp3" type="audio/mpeg">
      <a href="sounds/mp3/bass.mp3">Download bass.mp3</a>
    </audio>
  </article>
 </body>
</html>
```

Если открыть страницу в HTML5-совместимом браузере, каждый элемент списка будет представлен отдельным аудиопроигрывателем (рис. 7.1). Браузер обеспечивает воспроизведение аудио при нажатии кнопки `Play`.

Если открыть страницу в Internet Explorer, отображается ссылка для загрузки, так как браузер не понимает элемент `audio`. Даже это может стать неплохим обходным решением, но нельзя ли придумать что-нибудь получше?

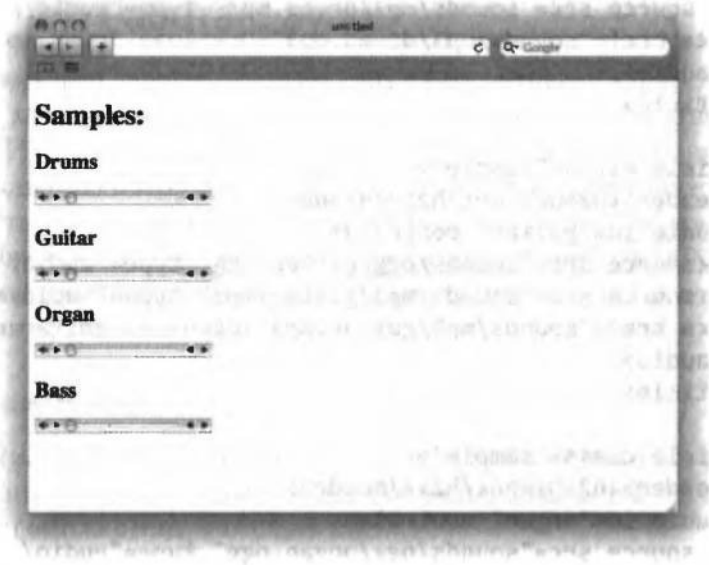


Рис. 7.1. Страница с семплами в Safari

Обходное решение

Обходная поддержка встроена в сам элемент `audio`. Мы определили несколько источников данных при помощи элемента `source`, а также предоставили ссылки для загрузки аудиофайлов. Если браузер не может воспроизвести элемент `audio`, он отображает внутреннюю ссылку. Можно было бы пойти еще дальше и использовать Flash после определения источников.

Тем не менее такое решение не идеально. Вам может попасться браузер, который поддерживает элемент `audio`, но не поддерживает заданные форматы. Или, допустим, вы решите, что предоставление аудио в нескольких форматах является лишней тратой времени. Кроме того, в спецификации HTML5 особо указано, что поддержка обходных решений для поддержки аудио не должна использоваться для размещения контента, читаемого экранными дикторами.

Простейшее решение — выведение ссылки для загрузки за пределы элемента `audio` и ее сокрытие средствами JavaScript.

`html5_audio/advanced_audio.html`

```
<article class="sample">
  <header><h2>Drums</h2></header>
```

```
  <audio id="drums" controls>
    <source src="sounds/ogg/drums.ogg" type="audio/ogg">
    <source src="sounds/mp3/drums.mp3" type="audio/mpeg">
  </audio>
  <a href="sounds/mp3/drums.mp3">Download drums.mp3</a>
</article>
```

Далее необходимо проверить поддержку аудио и скрыть ссылку в зависимости от ее результатов. Для этого мы создаем в JavaScript новый элемент `audio` и проверяем, реагирует ли он на вызов метода `canPlayType()`.

`html5_audio/advanced_audio.html`

```
var canPlayAudioFiles = !!(document.createElement('audio').
canPlayType);
```

В зависимости от того, какой ответ был получен, мы скрываем все якоря, вложенные в разделы семплов.

`html5_audio/advanced_audio.html`

```
$(function(){
  var canPlayAudioFiles = !!(document.createElement('audio').
canPlayType);

  if(canPlayAudioFiles){
    $(".sample a").hide();
  };
});
```

Обходные решения для работы с аудио реализуются относительно просто. Вероятно, некоторым из ваших пользователей даже понравится возможность просто загрузить нужный файл.

Встроенная поддержка воспроизведения аудио в браузере — всего лишь первый шаг. Браузеры еще только начинают поддерживать HTML5 JavaScript API для работы с аудио и видео (см. врезку на с. 148).

Рецепт 16. Внедрение видео

Фирма AwesomeCo хочет выложить на своем сайте новую серию учебных видеороликов, причем видеоролики должны просматриваться на как можно большем количестве устройств, особенно на iPad. В тестовых целях вам были предоставлены два видеоролика из серии «Уроки работы с Photoshop», которые должны использоваться для построения прототипа. К счастью, видеофайлы уже закодированы в форматах H.264, Theora и VP8, так что мы можем сосредоточиться на создании страницы[1].

Тег `video` практически полностью аналогичен элементу `audio`. Вы просто указываете источник, после чего Chrome, Firefox, Safari, iPhone, iPad и Internet Explorer 9 воспроизводят видео без каких-либо дополнительных плагинов. Разметка первого видеоролика *01_blur* выглядит следующим образом:

html5video/index.html

```
<article>
  <header>
    <h2>Saturate with Blur</h2>
  </header>
  <video controls>
    <source src="video/h264/01_blur.mp4">
    <source src="video/theora/01_blur.ogv">
    <source src="video/webm/01_blur.webm">
    <p>Your browser does not support the video tag.</p>
  </video>
</article>
```

Разметка определяет тег `video` с элементами управления воспроизведением. Отсутствие атрибута `autoplay` неявно указывает на то, что ролик не должен воспроизводиться автоматически.

Чтобы убедиться, что веб-браузер умеет обрабатывать видеофайлы, создадим новый .htaccess-файл в папке, где находится веб-страница, определяющая MIME-типы для видео.

Download html5video/.htaccess

```
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
```

---

[1] За дополнительной информацией о самостоятельном кодировании видеофайлов обращайтесь к приложению B.

После загрузки этих файлов в ваш веб-сервер видеоролик, оформленный таким образом, будет воспроизводиться в широком спектре браузеров. Примерный вид проигрывателя показан на рис. 7.2.

Это решение недоступно для пользователей Internet Explorer 8 и более ранних версий. Для воспроизведения видео в таких случаях приходится использовать Flash.

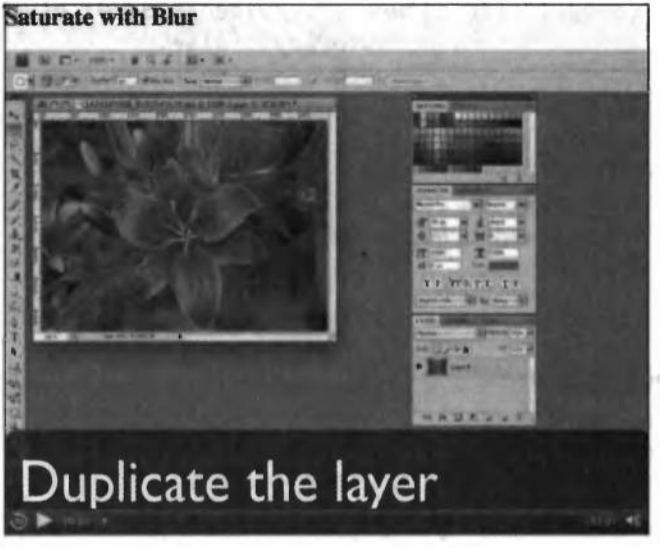


Рис. 7.2. Воспроизведение видеоролика в HTML5-проигрывателе Safari

Обходное решение

Чтобы правильно организовать обходное решение на базе Flash при использовании HTML5, мы включаем код объекта Flash в тег `video`. На сайте Video For Everybody[1] этот процесс описан исключительно подробно, а здесь будут представлены основные этапы реализации.

Flowplayer[2] — проигрыватель на базе Flash, который способен воспроизводить уже закодированное видео H.264. Загрузите версию проигрывателя с открытым кодом, разместите файлы *flowplayer-x.x.x.swf* и *flowplayer-controls-x.x.x.swf* в папке *swf* проекта.

Затем включите следующий фрагмент кода в тег `video`, сразу же за последним элементом `source`.

---

[1] http://videoforeverybody

[2] http://flowplayer.org/download/index.html

html5video/index.html

```
<object width="640" height="480"
type="application/x-shockwave-flash"
  data="swf/flowplayer-3.2.2.swf">
  <param name="movie" value="swf/flowplayer-3.2.2.swf" />
  <param name="allowfullscreen" value="true" />
  <param name="flashvars"
    value='config={"clip":{"url":"../video/h264/01_blur.mp4",
                           "autoPlay":false,
                           "autoBuffering":true
                          }
                }' />
  <img src="video/thumbs/01_blur.png"
    width="640" height="264" alt="Poster Image"
    title="No video playback capabilities." />
</object>
```

Обратите особое внимание на следующий фрагмент:

html5video/index.html

```
<param name="flashvars"
  value='config={"clip":{"url":"../video/h264/01_blur.mp4",
                         "autoPlay":false,
                         "autoBuffering":true
                        }
              }' />
```

Местонахождение источника видеофайла задается относительно местонахождения Flowplayer. Так как мы разместили Flowplayer в папке *swf*, для просмотра видеоролика следует указать путь *../video/h264/01_blur.mp4*.

Мы используем *самозакрывающиеся теги*. Как уже говорилось, обычно нет никакой необходимости использовать теги, недействующие в HTML5, например `<img>` и `<source>`, но существует ряд старых браузеров, которые не способны «прочитать» элемент `source`, а значит, никогда не покажут альтернативный контент. Самозакрывающиеся элементы `source` позволяют решить эту проблему.

При загрузке страницы в Internet Explorer видео успешно воспроизводится, а благодаря Flowplayer ролики не нужно кодировать в другой формат. Пользователи Internet Explorer видят проигрыватель, показанный на рис. 7.3.

Конечно, мы должны также обеспечить возможность просмотра видео людьми, браузеры которых не имеют встроенной поддержки `video` и у которых не установлена поддержка Flash. Чтобы такие пользователи могли загрузить видеоконтент, мы добавим еще один раздел со ссылками загрузки.

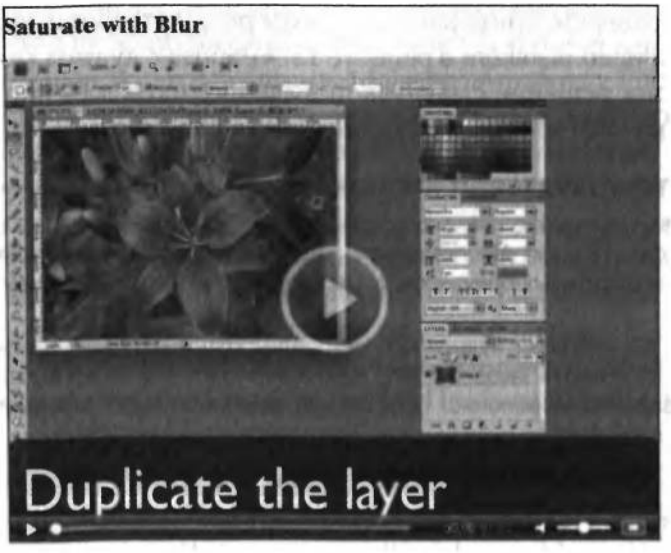


Рис. 7.3. Воспроизведение видео в Internet Explorer с использованием Flowplayer

html5video/index.html

```
<section class="downloads">
  <header>
    <h3>Downloads</h3>
  </header>
  <ul>
    <li><a href="video/h264/01_blur.mp4">H264, playable on most
    platforms</a></li>
    <li><a href="video/theora/01_blur.ogv">OGG format</a></li>
    <li><a href="video/webm/01_blur.webm">WebM format</a></li>
  </ul>
</section>
```

Если видео HTML5 не поддерживается, ссылки скрываются из кода JavaScript.

```
function canPlayVideo() {
  return !!document.createElement('video').canPlayType;
}
if(canPlayVideo()){
  $(#videos .downloads).hide();
}
```

Метод проверки, примененный в этом решении, очень похож на тот, который мы использовали в рецепте 15, «Работа с аудио» (с. 140). В нашей ситуации решение с загрузкой видеороликов для последующего просмотра на iPod или iPad выглядит более разумно.

MEDIA CONTENT JAVASCRIPT API

В этой главе мы кратко познакомились с JavaScript API для работы с элементами audio и video. Полный API позволяет получать информацию о типах аудиофайлов, воспроизводимых браузеров, а также управлять воспроизведением элементов audio.

В рецепте 15, «Работа с аудио» мы построили страницу с несколькими звуковыми семплами. Используя JavaScript API, можно заставить все звуки воспроизводиться (практически) одновременно. Простейшее решение может выглядеть примерно так:

Download html5_audio/advanced_audio.html

```
var element = $("<p><input type='button' value='Play all'/></
p>")
element.click(function(){
  $("audio").each(function(){
    this.play();
  })
});

$("body").append(element);
```

При нажатии кнопки «Play All» этот фрагмент перебирает все элементы audio и вызывает метод play() для каждого элемента.

Аналогичным образом можно управлять и воспроизведением видео. В API имеются методы запуска и приостановки воспроизведения и даже получения информации о текущей позиции.

К сожалению, на момент написания книги JavaScript API недостаточно хорошо поддерживается некоторыми браузерами. Несмотря на это, вам определенно стоит познакомиться со всеми возможностями по спецификации[1].

[1] http://www.w3.org/TR/html5/video.html#media-elements

Ограничения поддержки видео в HTML5

В настоящее время полезность поддержки видео в HTML5 ограничивается тремя очень важными обстоятельствами.

Во-первых, видео HTML5 не предусматривает потоковой передачи видеофайлов. Пользователи уже привыкли к возможности позиционирования к конкретной части видеофайла. В этой области особенно хорошо проявляют себя видеопроигрыватели на базе Flash (благодаря огромным усилиям, которые затратила фирма Adobe на разработку Flash как платформы доставки видеоконтента). Для выполнения позиционирования в видеоролике HTML5 файл необходимо полностью загрузить в браузере (хотя, возможно, со временем ситуация изменится).

Во-вторых, в нем не поддерживаются механизмы управление правами. Такие сайты, как Hulu[1], желающие предотвратить пиратское использование своего контента, не могут положиться на видео HTML5. Flash остается эффективным решением для подобных ситуаций.

В-третьих (что самое важное), процесс кодирования видео требует значительных затрат времени и ресурсов. Необходимость кодирования в нескольких форматах снижает привлекательность решений HTML5. По этой причине многие сайты поставляют видео в формате H.264 со всеми его патентными сложностями, чтобы оно могло воспроизводиться на iPod и iPad, с использованием комбинации тега HTML5 `video` и Flash.

Эти проблемы не лишают HTML5 перспектив, однако вы должны учитывать их, прежде чем принимать решение о переходе с Flash на HTML5 в качестве технологии передачи видео.

ПОРНО КАК ДВИГАТЕЛЬ ПРОГРЕССА

Отрасль интимных развлечений оказала серьезное влияние на развитие многих интернет-технологий, от электронной коммерции до Flash[2], и продолжает влиять на технологии работы с видео HTML5[3]. Такие устройства, как iPhone и iPad, имеют более «приватный» характер, чем настольные и портативные компьютеры, и Flash на них не работает. Многие сайты «для взрослых» по этой причине уже начали переводить свой видеоконтент с Flash на HTML5 с видео H.264. Интересно, что их пока не слишком беспокоит отсутствие управления правами в технологиях видео HTML5.

Эта отрасль не боится рисковать. Возможно, проявленный интерес будет способствовать новым достижениям в области технологий видео HTML5.

---

[1] http://www.hulu.com

[2] http://chicagopressrelease.com/news/in-tech-world-porn-quietly-leads-the-way

[3] http://news.avn.com/articles/Joone-Points-to-HTML-5-as-Future-of-Web-Content-Delivery-401434.html

Аудио, видео и доступность

Ни одно из обходных решений не обладает особой эффективностью для пользователей с физическими недостатками. Более того, на этот факт явно указано в спецификации HTML5. Возможность загрузки файла не решит проблем пользователя с недостатками слуха, а пользователю с недостатками зрения не нужен видеофайл, который должен просматриваться вне браузера. Предоставляя контент пользователям, следует по возможности предусмотреть разумные альтернативы. Видео- и аудиофайлы должны иметь текстовые расшифровки, которые могут просматриваться пользователями. Если вы производите собственный контент, то планирование расшифровки с самого начала позволяет легко создать ее, потому что текст берется из написанного сценария. Если полная расшифровка невозможна, рассмотрите возможность создания сводки с описанием важнейших частей видеосюжета.

html5video/index.html

```
<section class="transcript">
  <h2>Transcript</h2>
  <p>We'll drag the existing layer to the new button on the
     bottom of the Layers palette to create a new copy.</p>
  <p>Next we'll go to the Filter menu and choose "Gaussian
     Blur". We'll change the blur amount just enough so that we
     lose a little bit of the detail of the image.</p>
  <p>Now we'll double-click on the layer to edit the layer and
     change the blending mode to "Overlay". We can then adjust
     the amount of the effect by changing the opacity slider.</p>
  <p>Now we have a slightly enhanced image.</p>
</section>
```

Расшифровку можно скрыть или же разместить ссылку на нее на основной странице с видео. Если у пользователя не будет проблем с ее поиском и переходом, такая возможность будет чрезвычайно полезной.

Перспективы

Встроенная поддержка аудио браузерами открывает множество новых возможностей перед разработчиками. Веб-приложения JavaScript могут легко выдавать звуковые эффекты и сигналы без встраивания аудиокон-

тента средствами Flash. Встроенная поддержка видео позволяет сделать его доступным на таких устройствах, как iPhone, а также предоставляет открытый, стандартный метод взаимодействия с видео из кода JavaScript. Но самое важное, что с видео- и аудиоклипами можно поступать точно так же, как мы поступаем с графикой — снабжать их семантической разметкой для упрощения идентификации.

8 Визуальные эффекты

Веб-разработчики всегда ищут возможности сделать интерфейс пользователя более привлекательным. CSS3 предоставляет для этого много новых возможностей. Вы можете использовать собственные шрифты на страницах; создавать элементы с закругленными углами и тенями; использовать градиенты в качестве фона, и даже поворачивать элементы, чтобы они не казались такими однообразными и геометрически правильными. Все это делается без применения Photoshop и других графических программ; в этой главе мы покажем, что для этого нужно сделать. Для начала вы узнаете, как закруглить острые углы на форме. Затем мы построим баннер для предстоящей торгово-промышленной выставки, и вы научитесь добавлять тени, повороты, градиенты и прозрачность. Глава завершается описанием новой возможности CSS3 `@font-face`, позволяющей использовать более симпатичные шрифты в блоге компании.

В этой главе рассматриваются следующие возможности CSS3[1]:

`border-radius` [`border-radius: 10px;`]

Закругление углов элементов. [C4, F3, IE9, S3.2, O10.5]

Поддержка RGBa [`background-color: rgba(255,0,0,0.5);`]

Использование цветов RGB вместо шестнадцатеричных кодов (с альфа-каналом) [C4, F3.5, IE9, S3.2, O10.1]

---

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

`box-shadow` [`box-shadow: 10px 10px 5px #333;`]

Создание теней, отбрасываемых элементами.[C3, F3.5, IE9, S3.2, O10.5]

Повороты [`transform: rotate(7.5deg);`]

Поворот любого элемента. [C3, F3.5, IE9, S3.2, O10.5]

Градиенты [`linear-gradient(top, #fff, #efefef);`]

Построение градиента для использования в качестве графического изображения. [C4, F3.5, S4]

`@font-face` [`@font-face { font-family: AwesomeFont; ] src: url(http://example.com/awesomeco.ttf); font-weight: bold; }`]

Возможность использования конкретных шрифтов в CSS. [C4, F3.5, IE5+, S3.2, O10.1]

Рецепт 17. Закругление прямых углов

В Web все элементы по умолчанию имеют прямоугольную форму. Поля форм, таблицы и даже разделы веб-страниц — все они имеют строгий, геометрически правильный вид. За прошедшие годы многие веб-дизайнеры использовали различные способы закругления углов элементов, чтобы немного смягчить интерфейс. В CSS3 появилась поддержка простого и удобного закругления углов, которая уже довольно давно поддерживается в Firefox и Safari. К сожалению, Internet Explorer пока отстает от них, но эту проблему можно легко обойти.

Закругленные углы на форме входа

На макетах, полученных вами для текущего проекта, изображены поля форм с закругленными углами. Начнем с реализации этого эффекта исключительно средствами CSS3. Примерный вид того, что нужно сделать, показан на рис. 8.1.

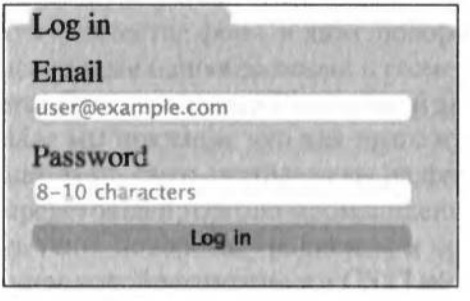


Рис. 8.1. Форма с закругленными углами

Форма входа в систему реализуется очень простым кодом HTML.

css3roughedges/rounded_corners.html

```
<form action="/login" method="post">
  <fieldset id="login">
    <legend>Log in</legend>
    <ol>
      <li>
        <label for="email">Email</label>
        <input id="email" type="email" name="email">
      </li>
      <li>
```

```
        <label for="password">Password</label>
        <input id="password" type="password"
               name="password" value="" autocomplete="off"/>
      </li>
      <li><input type="submit" value="Log in"></li>
    </ol>
  </fieldset>
</form>
```

Применение к форме стилей немного улучшит ее внешний вид.

css3roughedges/style.css

```
fieldset{
  width: 216px;
  border: none;
  background-color: #ddd;
}

fieldset legend{
  background-color: #ddd;
  padding: 0 64px 0 2px;
}

fieldset>ol{list-style: none;
  padding:0;
  margin: 2px;
}

fieldset>ol>li{
  margin: 0 0 9px 0;
  padding: 0;
}

/* Текстовые поля выводятся с новой строки */
fieldset input{
  display:block;
}

input{
  width: 200px;
  background-color: #fff;
  border: 1px solid #bbb;
}
```

```
input[type="submit"]{
  width: 202px;
  padding: 0;
  background-color: #bbb;
}
```

Эти стили удаляют из списка маркеры и обеспечивают равенство размеров текстовых полей. Когда базовое оформление будет завершено, можно переходить к применению эффектов закругления.

Селекторы, поддерживаемые отдельными браузерами

Так как спецификация CSS3 еще не является окончательной, фирмы-разработчики браузеров добавили в свои продукты некоторые возможности на свое усмотрение. Такие реализации снабжаются префиксами; это позволяет разработчикам браузеров вводить новую функциональность еще до того, как она будет закреплена в окончательной спецификации. А поскольку эти реализации не соответствуют фактической спецификации, разработчик браузера может позднее реализовать официальную спецификацию с сохранением собственной реализации. Обычно реализация с префиксом совпадает с реализацией из спецификации CSS, но время от времени встречаются различия. Неприятное последствие для вас означает, что радиус закругления придется объявлять отдельно для каждого браузера.

В Firefox используется следующий селектор:

css3roughedges/style.css

```
-moz-border-radius: 5px;
```

У браузеров на базе WebKit (такие, как Safari и Chrome) селектор выглядит немного иначе.

css3roughedges/style.css

```
-webkit-border-radius: 5px;
```

Для закругления углов всех текстовых полей формы понадобится правило CSS вида

css3roughedges/style.css

```
input, fieldset, legend{
  border-radius: 5px;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
}
```

Включите его в файл *style.css*, — и углы текстовых полей будут плавно закругляться.

Обходное решение

Приведенное решение работает в Firefox, Safari и Google Chrome, но мы знаем, что в Internet Explorer оно работать не будет, а следовательно, нужно реализовать как можно более близкий аналог. До настоящего времени веб-разработчики реализовали эффект закругления углов при помощи фоновых изображений и других приемов, но мы постараемся использовать как можно более простое решение. Радиус можно определить из JavaScript, после чего для закругления углов применяется один из многих существующих методов. В нашем примере будет использоваться jQuery, плагин jQuery Corner и модификация плагина Corner для закругления углов текстовых полей.

Проверка поддержки закругления

Наше обходное решение очень похоже на то, которое использовалось в разделе 9. Мы включаем библиотеку jQuery и плагин, проверяем, поддерживает ли браузер наш атрибут, и если не поддерживает — активируем плагин. В данном случае необходимо проверить присутствие свойства CSS `border-radius` и префиксы конкретных браузеров (такие, как `webkit` и `moz`).

Создайте файл *corner.js* и включите в него следующую функцию:

css3roughedges/corner.js

```
function hasBorderRadius(){
  var element = document.documentElement;
  var style = element.style;
  if (style){
    return typeof style.borderRadius == "string" ||
      typeof style.MozBorderRadius == "string" ||
      typeof style.WebkitBorderRadius == "string" ||
      typeof style.KhtmlBorderRadius == "string";
  }
  return null;
}
```

Теперь, когда мы знаем, как проверить браузер на поддержку закругления углов, мы напишем код для выполнения собственно закругления. К счастью, существует специальный плагин, который поможет нам

СТОИТ ЛИ ИГРА СВЕЧ!

В нашем примере клиент требует, чтобы во всех браузерах углы были закруглены. Однако такие возможности всегда следует оставлять на усмотрение пользователя. Хотя кто-то скажет, что «смягчение» внешнего вида формы приносит реальную пользу, для начала необходимо прикинуть, сколько пользователей работает в браузерах без поддержки закругления средствами CSS. Если ваши посетители в основном работают в Safari и Firefox, возможно, написание и сопровождение сценария проверки и реализации обходного решения будет оправдано.

Плагин jQuery Corners

jQuery Corners[1] — небольшой плагин для закругления углов прямоугольных элементов. Для этого элемент упаковывается в дополнительные теги `div`, и к нему применяется стилевое оформление, имитирующее эффект закругления. С полями форм он не работает, однако использование этого плагина в сочетании с jQuery позволит добиться желаемого результата.

Загрузите плагин jQuery Corners и подключите его к странице HTML. Заодно подключите файл *corner.js*.

`css3roughedges/rounded_corners.html`

```
<script src="jquery.corner.js" charset="utf-8" type='text/
javascript'></script>
<script src="corner.js" charset="utf-8" type='text/
javascript'></script>
```

Теперь мы напишем код, непосредственно выполняющий округление.

Плагин formCorners

Мы напишем плагин jQuery, который будет легко применять закругление ко всем полям формы. Написание плагинов jQuery уже рассматривалось в рецепте 5 (с. 60). Мы не будем повторяться, а проанализируем код плагина, частично базирующийся на решении Тони Амояла[2].

[1] http://www.malsup.com/jquery/corner/

[2] http://www.tonyamoyal.com/2009/06/23/text-inputs-with-rounded-corners-using-jquery-without-image/

Включите в файл *corners.js* следующий фрагмент:

css3roughedges/corner.js

```
(function($){

  $.fn.formCorner = function(){
    return this.each(function() {
      var input = $(this);
      var input_background = input.css("background-color");
      var input_border = input.css("border-color");
      input.css("border", "none");
      var wrap_width = parseInt(input.css("width")) + 4;
      var wrapper = input.wrap("<div></div>").parent();
      var border = wrapper.wrap("<div></div>").parent();
      wrapper.css("background-color", input_background)
             .css("padding", "1px");
      border.css("background-color",input_border)
            .css("width", wrap_width + "px")
            .css('padding', '1px');
      wrapper.corner("round 5px");
      border.corner("round 5px");
    });
  };
})(jQuery);
```

Мы берем объект jQuery, который может представлять элемент или коллекцию элементов, и заключаем его в два тега `div`, к которым применяется закругление. Внутреннему тегу `div` назначается цвет фона исходного поля ввода, после чего мы отключаем обрамление исходного поля формы. Это поле заключается в другое поле со своим цветом фона, совпадающим с цветом рамки исходного поля, для которого назначаются отступы (padding). Именно отступы делают видимым контур обрамления. Представьте два куска картона — зеленый шириной в 10 см и красный шириной в 7 см. Наложив меньший кусок на больший, вы увидите красное поле с зеленой рамкой. Этот принцип заложен в основу всего решения.

Активизация закругления

Когда плагин и библиотека проверки будут готовы, можно переходить к активизации механизма закругления. Включите в файл *corners.js* следующий фрагмент:

css3roughedges/corner.js

```
1 $(function(){
2   if(!hasBorderRadius()){
3     $("input").formCorner();
4     $("fieldset").corner("round 5px");
5     $("Legend").corner("round top 5px cc:#fff");
6   }
7 });
```

Закругление применяется к трем полям формы и группе полей (fieldset); в строке 5 мы применяем закругление только к верхней части заголовка группы и указываем, что в вырезе угла должен использоваться белый цвет. Плагин использует в качестве цвета выреза цвет фона родителя, что в данной ситуации неуместно.

Если браузер поддерживает свойство `border-radius`, то выполняется наш плагин. Если нет — используется добавленный ранее код CSS.

Небольшое смещение

В IE заголовки групп полей прорисовываются немного иначе. Мы добавим небольшую «заплатку» для IE, который слегка смещает заголовки вверх, чтобы результат выглядел так же, как в Firefox и Chrome.

css3roughedges/rounded_corners.html

```
<link rel="stylesheet" href="style.css" type="text/css"
media="screen">
<!--[if IE]>
  <style>
    fieldset legend{margin-top: -10px }
  </style>
<![endif]-->
```

Теперь во всех основных браузерах результат выглядит одинаково; версия для Internet Explorer показана на рис. 8.2.

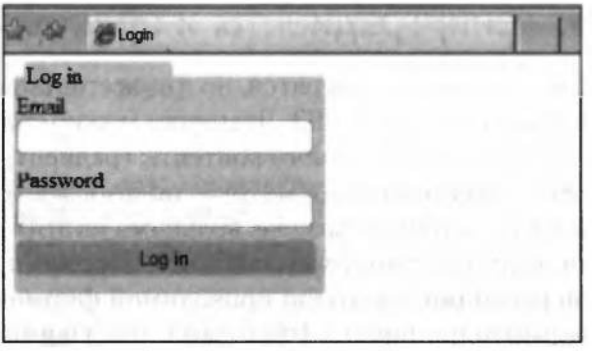


Рис. 8.2. Форма с закругленными углами в Internet Explorer

Закругленные углы визуально «смягчают» интерфейс, а использовать их чрезвычайно легко. Вместе с тем, очень важно последовательно подходить к их применению и не злоупотреблять этим аспектом дизайна (как, впрочем, и любым другим).

Рецепт 18. Тени, градиенты и преобразования

Закругленные углы хорошо смотрятся, но это всего лишь начало того, что можно сделать средствами CSS3. Элементы можно снабдить тенями, выделяющими их на фоне остального контента; градиенты придают фонам элегантность; преобразования могут использоваться для поворота элементов. Мы воспользуемся сразу несколькими из этих возможностей для построения макета баннера AwesomeConf — торгово-промышленной выставки и конференции, ежегодно проводимой фирмой AwesomeCo. Художник-оформитель прислал PSD-файл, показанный на рис. 8.3. Карточка с именем, тень и даже прозрачность — все эти возможности могут быть реализованы исключительно на уровне CSS. От художника нам понадобится только одно: фоновое изображение людей.

Рис. 8.3. Исходный макет, который мы воссоздадим средствами CSS3

Основная структура

Начнем с разметки базовой структуры страницы в HTML.

css3banner/index.html

```
<div id="conference">
  <section id="badge">
    <h3>Hi, My Name Is</h3>
    <h2>Barney</h2>
  </section>

  <section id="info">
  </section>
</div>
```

Отправной точкой для стилевого оформления базовой структуры станет следующая страница:

css3banner/style.css

```
#conference{
  background-color: #000;
  width: 960px;
  float:left;
  background-image: url('images/awesomeconf.jpg');
  background-position: center;
  height: 240px;
}
#badge{
  text-align: center;
  width: 200px;
  border: 2px solid blue;
}

#info{
  margin: 20px;
  padding: 20px;
  width: 660px;
  height: 160px;
}

#badge, #info{
  float: left;
  background-color: #fff;
}

#badge h2{
  margin: 0;
  color: red;
  font-size: 40px;
}

#badge h3{
  margin: 0;
  background-color: blue;
  color: #fff;
}
```

После применения этой таблицы стилей к странице карточка с именем и область контента отображаются рядом друг с другом (рис. 8.4). Начнем со стилевого оформления карточки с именем.

Рис. 8.4. Исходная версия баннера

Добавление градиента

Чтобы карточка с именем смотрелась более профессионально, мы заменим белый фон градиентом, переходящим от белого к светло-серому. Градиенты работают в Firefox, Safari и Chrome, но реализация в Firefox несколько отличается от других. В Chrome и Safari реализован синтаксис WebKit, содержавшийся в исходном предложении, тогда как в Firefox реализован синтаксис, близкий к предложению W3C. Как и в предыдущих случаях, мы используем префиксы конкретных браузеров[1] (см. рецепт 17, с. 154).

css3banner/style.css

```
#badge{
  background-image: -moz-linear-gradient(
    top, #fff, #efefef
  );

  background-image: -webkit-gradient(
    linear,left top, left bottom,
    color-stop(0, #fff),
    color-stop(1, #efefef)
  );

  background-image: linear-gradient(
    top, #fff, #efefef
  );
}
```

Firefox использует метод `-moz-linear-gradient`, в параметрах которого указывается начальная точка градиента, начальный и конечный цвета.

[1] http://dev.w3.org/csswg/css3-images/#linear-gradients

Браузеры на базе WebKit позволяют задавать промежуточные цвета. В нашем примере градиент переходит от белого к серому, но при необходимости в определение можно добавить дополнительные переходы.

Добавление тени

Тень, отбрасываемая карточкой с именем, визуально «приподнимает» ее над баннером. Традиционно для реализации теней использовался редактор Photoshop — тень либо добавлялась в изображение, либо вставлялась в виде фонового изображения. Новое свойство CSS3 `box-shadow` позволяет быстро определять тени для элементов[1]. Применение следующего правила к таблице стилей создает тень у карточки с именем:

`css3banner/style.css`

```
#badge{
  -moz-box-shadow: 5px 5px 5px #333;
  -webkit-box-shadow: 5px 5px 5px #333;
  -o-box-shadow: 5px 5px 5px #333;
  box-shadow: 5px 5px 5px #333;
}
```

Свойство `box-shadow` имеет четыре параметра. Первый параметр определяет горизонтальное смещение. При положительном значении тень располагается справа от элемента, а при отрицательном — слева. Второй параметр определяет вертикальное смещение. С положительными значениями этого параметра тень располагается выше элемента, а при отрицательных — ниже.

ТЕНИ И ТЕКСТ

Тени могут назначаться не только визуальным элементам, но и тексту. Такие тени используются практически так же, как и тени box-shadow:

```
h1{text-shadow: 2px 2px 2px #bbbbbb;}
```

Вы указываете смещения X и Y, величину размывки и цвет тени. В IE 6, 7 и 8 для поддержки текстовых теней используется фильтр Shadow:

```
filter: Shadow(Color=#bbbbbb,
Direction=135,
Strength=3);
```

Тени, отбрасываемые текстом, эффектно смотрятся, но при чрезмерной контрастности они затрудняют чтение.

[1] http://www.w3.org/TR/css3-background/#the-box-shadow

Третий параметр определяет радиус размывки. Со значением 0 тень выглядит очень контрастно, а с более высокими значениями она размывается. Последний параметр определяет цвет тени.

Поэкспериментируйте с параметрами, получите представление о том, как они работают, и найдите значения, которые кажутся вам наиболее подходящими. Выделите немного времени на изучение теней в реальном мире. Возьмите фонарик и посветите им на какие-нибудь предметы или понаблюдайте за тем, как солнце отбрасывает тени на объекты. Перспектива играет важную роль, потому что непоследовательное применение тени только собьет с толку пользователей (особенно если тени неправильно назначаются сразу нескольким элементам). Самое простое решение — использовать одни и те же параметры для всех создаваемых теней.

Поворот карточки с именем

Преобразования CSS3 могут использоваться для поворота, масштабирования и деформации элементов — по аналогии с операциями, выполняемыми в программах векторной графики (таких, как Flash, Illustrator или Inkscape)[1]. Элементы становятся более рельефными, а веб-страница перестает выглядеть плоской и однообразной. Давайте немного повернем карточку, чтобы она выходила за границы контура баннера.

css3banner/style.css

```
.badge{
  -moz-transform: rotate(-7.5deg);
  -o-transform: rotate(-7.5deg);
  -webkit-transform: rotate(-7.5deg);
  -ms-transform: rotate(-7.5deg);
  transform: rotate(-7.5deg);
```

Повороты в CSS3 выполняются достаточно просто. Вы лишь указываете угол в градусах, а браузер делает все остальное. Вместе с элементом поворачивается все его содержимое.

Поворот выполняется так же просто, как и закругление углов, однако и тоже не стоит злоупотреблять. Интерфейс, прежде всего, должен быть удобным в использовании. Если вы поворачиваете элементы, содержащие большой объем контента, убедитесь в том, что пользователи смогут нормально прочитать контент и им не придется наклонять голову!

---

[1] http://www.w3.org/TR/css3-2d-transforms/#transform-property

Прозрачные фоны

Веб-разработчики уже давно используют полупрозрачные слои за текстом в своей работе. Обычно для этого им приходилось либо создавать полное изображение в Photoshop, либо накладывать прозрачное изображение PNG поверх другого элемента средствами CSS. CSS3 позволяет определять фоновые цвета с новым синтаксисом, поддерживающим прозрачность.

При первом знакомстве с веб-программированием разработчик учится определять цвета в шестнадцатеричных кодах. Интенсивность красной, зеленой и синей цветовых составляющих задается двумя шестнадцатеричными цифрами: 00 означает полное отсутствие, а FF — максимальную интенсивность цвета. Таким образом, красный цвет кодируется последовательностью FF0000, то есть «красный на максимуме, зеленого нет, синего нет».

В CSS3 появились функции `rgb` и `rgba`. Функция `rgb` близка к своему шестнадцатеричному аналогу, но каждый цвет кодируется значениями от 0 до 255. Так, красный цвет определяется в виде `rgb(255,0,0)`.

Функция `rgba` работает аналогичным образом, но получает четвертый параметр, определяющий уровень прозрачности (от 0 до 1). Если параметр равен 0, то цвет вообще не виден, потому что он полностью прозрачен. Чтобы белый прямоугольник стал полупрозрачным, мы добавляем следующее стилевое правило:

css3banner/style.css

```
#info{
  background-color: rgba(255,255,255,0.95);
}
```

При использовании прозрачности настройки контраста, заданные пользователем, иногда могут влиять на внешний вид страницы; обязательно поэкспериментируйте со значением, проверьте результат на разных экранах и убедитесь в его стабильности.

Раз уж мы занялись информационной частью баннера, давайте немного закруглим углы.

css3banner/style.css

```
#info{
  moz-border-radius: 12px;
  webkit-border-radius: 12px;
  o-border-radius: 12px;
  border-radius: 12px;
}
```

С таким оформлением баннер хорошо смотрится в Safari, Firefox и Chrome. Остается реализовать таблицу стилей для Internet Explorer.

Обходное решение

Приемы, использованные в этом разделе, отлично работают в IE9, но все они также возможны в Internet Explorer 6, 7 и 8! Однако для их реализации необходимы фильтры Microsoft DirectX, а это означает, что в страницу необходимо включить условный комментарий для загрузки таблицы стилей, предназначенной исключительно для IE. Также для создания элемента `section`, который будет оформляться средствами CSS, придется использовать JavaScript, так как в этих версиях IE отсутствует встроенная поддержка этого элемента.

css3banner/index.html

```
    <!--[if lte IE 8]>

      <script>
      document.createElement("section");
      </script>

      <link rel="stylesheet" href="ie.css"
      type="text/css" media="screen">

    <![endif]-->
  </head>
  <body>
    <div id="conference">
      <section id="badge">
        <h3>Hi, My Name Is</h3>
        <h2>Barney</h2>
      </section>

      <section id="info">
      </section>
    </div>

  </body>
</html>
```

Фильтры DirectX работают в IE6, 7 и 8, но в IE8 они активизируются несколько иначе, поэтому каждый из фильтров приходится объявлять дважды.

Давайте посмотрим, как выполняется поворот элементов.

Поворот

Фильтры могут использоваться для поворота элементов, но для этого недостаточно указать угол поворота. Чтобы добиться желаемого результата, необходимо использовать фильтр Matrix и задать синусы и косинусы нужного угла. А если говорить точнее, нужно передать косинус, отрицательное значение синуса, синус и снова косинус[1]:

css3banner/filters.css

```
filter: progid:DXImageTransform.Microsoft.Matrix(
  sizingMethod='auto expand',
    M11=0.9914448613738104,
    M12=0.13052619222005157,
    M21=-0.13052619222005157,
    M22=0.9914448613738104
  );

-ms-filter: "progid:DXImageTransform.Microsoft.Matrix(
  sizingMethod='auto expand',
    M11=0.9914448613738104,
    M12=0.13052619222005157,
    M21=-0.13052619222005157,
    M22=0.9914448613738104
  )";
```

Сложно? Да, и становится еще сложнее, если присмотреться к примеру повнимательнее. Вспомните, что исходный угол поворота 7,5° был *отрицательным*. Таким образом, «отрицательный» синус будет иметь положительное значение, а собственно синус будет иметь отрицательное значение.

Ох уж эта математика... Переходим к градиентам.

Градиенты

Фильтр Gradient в IE работает так же, как стандартный, не считая того, что вам придется вводить намного больше символов. Вы задаете начальный и конечный цвета, а градиент строится автоматически.

css3banner/filters.css

```
filter: progid:DXImageTransform.Microsoft.gradient(
    startColorStr=#FFFFFF, endColorStr=#EFEFEF
);
```

[1] Линейное преобразование с матрицей 2×2.

```
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(
    startColorStr=#FFFFFF, endColorStr=#EFEFEF
)";
```

В отличие от других браузеров, градиент применяется непосредственно к элементу, а не к свойству `background-image`.

Мы снова воспользуемся этим фильтром для определения прозрачного фона раздела `info`.

Прозрачность

Фильтр Gradient получает расширенные шестнадцатеричные значения начального и конечного цветов; первые две цифры определяют уровень прозрачности. Следующий фрагмент кода реализует эффект, очень близкий к тому, что мы хотим получить.

`css3banner/filters.css`

```
background: none;
filter:
  progid:DXImageTransform.Microsoft.gradient(
  startColorStr=#BBFFFFFF, endColorStr=#BBFFFFFF
);

-ms-filter: "progid:DXImageTransform.Microsoft.gradient(
  startColorStr='#BBFFFFFF', EndColorStr='#BBFFFFFF'
)";
```

Шестнадцатеричные коды из восьми цифр очень похожи на функцию `rgba`, разве что уровень прозрачности указывается в начале, а не в конце. Таким образом, перечисляются уровни прозрачности, красного, зеленого и синего цвета.

Чтобы это решение работало в IE7, необходимо удалить свойства фона для этого элемента. Если вы следили за описанием, параллельно создавая собственную таблицу стилей, то заметили, что наше решение не работает, однако проблему можно решить.

Все вместе

Одна из проблем при использовании фильтров IE заключается в том, что они не могут определяться «по частям». Чтобы применить несколько фильтров к одному элементу, необходимо определить их в виде списка, разделенного запятыми. Вот как будет выглядеть реальная таблица стилей IE.

css3banner/ie.css

```
#info{
  background: none;
  filter:
    progid:DXImageTransform.Microsoft.gradient(
    startColorStr=#BBFFFFFF, endColorStr=#BBFFFFFF
  );
  -ms-filter: "progid:DXImageTransform.Microsoft.gradient(
    startColorStr='#BBFFFFFF', EndColorStr='#BBFFFFFF'
  )";
}

#badge{
  filter:
    progid:DXImageTransform.Microsoft.Matrix(
      sizingMethod='auto expand',
      M11=0.9914448613738104,
      M12=0.13052619222005157,
      M21=-0.13052619222005157,
      M22=0.9914448613738104
    ),
    progid:DXImageTransform.Microsoft.gradient(
      startColorStr=#FFFFFF, endColorStr=#EFEFEF
    ),
    progid:DXImageTransform.Microsoft.Shadow(
      color=#333333, Direction=135, Strength=3
    );

  -ms-filter: "progid:DXImageTransform.Microsoft.Matrix(
    sizingMethod='auto expand',
    M11=0.9914448613738104,
    M12=0.13052619222005157,

   M21=-0.13052619224147205157,
    M22=0.99144486137384
  ),

  progid:DXImageTransform.Microsoft.gradient(
    startColorStr=#FFFFFF, endColorStr=#EFEFEF
  ),

  progid:DXImageTransform.Microsoft.Shadow(
    color=#333333, Direction=135, Strength=3
  )";
}
```

Код получается довольно объемистым, но он наглядно показывает, что этими возможностями можно пользоваться. Посмотрите на рис. 8.5 — результат довольно близок к цели. Осталось лишь закруглить углы раздела `info`; о том, как это делается, рассказано на с. 154.

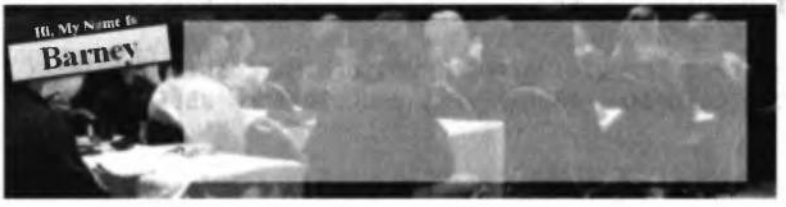


Рис. 8.5. Баннер AwesomeCo в Internet Explorer 8

Фильтры громоздки, к их особенностям привыкаешь не сразу — и все же вам стоит познакомиться с ними поближе, чтобы вы могли реализовать аналогичную функциональность в собственных проектах для пользователей IE.

Все эффекты, рассмотренные в этом разделе, относятся исключительно к визуальному представлению. Создавая исходную таблицу стилей, мы не забыли применить цвет фона, обеспечивающий нормальную читаемость текста в браузерах, не поддерживающих синтаксис CSS3.

Рецепт 19. Использование шрифтов

Подбор шрифтов является важной составляющей впечатлений пользователя от работы на сайте. Например, шрифты в той книге, которую вы сейчас читаете, были тщательно отобраны людьми, которые хорошо знают, как правильный выбор шрифтов и интервалов упрощает восприятие текста. Эти концепции играют не менее важную роль и в дизайне веб-страниц.

Шрифты, выбираемые для передачи информации пользователю, влияют на интерпретацию этой информации. Этот шрифт идеально подойдет для сайта группы, играющей громкую музыку в стиле «хэви-метал».

С другой стороны, он вряд ли будет уместен для оформления темы этой книги.

Как видите, очень важно выбрать шрифт, соответствующий информационному наполнению страницы. Традиционно выбор веб-разработчиков ограничивался небольшим подмножеством так называемых «веб-безопасных» шрифтов, встречающихся в операционных системах подавляющего большинства пользователей.

Для решения этой проблемы приходилось использовать графику — либо включать изображения непосредственно в разметку страницы, либо использовать другие способы: фоновые изображения CSS или sIFR[1] с отображением шрифтов на базе Flash.

Модуль CSS3 Fonts предоставляет намного более удобное решение.

---

[1] http://www.mikeindustries.com/blog/sifr

@font-face

Вообще говоря, директива `@font-face` появилась в спецификации CSS2 и была реализована еще в Internet Explorer 5. Однако в реализации Microsoft использовался формат шрифтов EOT (Embedded OpenType), а большинство современных шрифтов хранится в формате TrueType или OpenType. В настоящее время другие браузеры поддерживают шрифты OpenType и TrueType.

ШРИФТЫ И ПРАВА

Далеко не все шрифты распространяются бесплатно. Их использование, как и использование готовых фотографий и других материалов, защищенных авторским правом, должно соответствовать авторским правам и условиям лицензирования для вашего сайта. Приобретая шрифт, вы обычно получаете право использовать его в логотипах и графике на странице (это называется «правами использования»). Однако при использовании @font-face приходится учитывать новую разновидность лицензирования — «права повторного распространения».

Когда вы встраиваете шрифт в свою веб-страницу, вашим пользователям придется его загружать; таким образом, ваш сайт начинает распространять шрифт среди других пользователей. Вы должны быть абсолютно уверены в том, что шрифты, используемые на страницах, допускают такой тип использования.

Typekit[1] предоставляет большую библиотеку лицензированных шрифтов, а также инструменты и код, упрощающие их интеграцию на сайтах. Сервис не бесплатный, но расценки вполне приемлемые, если вы хотите использовать конкретный шрифт.

Google предоставляет прикладной интерфейс Google Font API[2], сходный с Typekit, но использующий только свободно распространяемые шрифты.

Оба вида сервиса используют JavaScript для загрузки шрифтов. Следовательно, вы должны позаботиться о том, чтобы контент нормально читался у пользователей с отключенной поддержкой JavaScript.

Помните, что к шрифтам следует относиться так же, как и к любому другому ресурсу, — и у вас не будет никаких проблем.

Директор по маркетингу фирмы AwesomeCo решил, что компания должна стандартизировать использование шрифтов в печатной продукции и на сайте. Вам было предложено разобраться с Garogier — простым шрифтом, полностью бесплатным для коммерческого использования. Для пробы мы используем этот шрифт в примере блога, созданном в рецепте 1, «Реструктуризация блога с использованием семантической разметки» на с. 28, чтобы все желающие смогли увидеть, как выглядит этот шрифт на практике.

[1] http://www.typekit.com/

[2] http://code.google.com/apis/webfonts/

ВОПРОС\ОТВЕТ

Как мне преобразовать мои собственные шрифты?

Если вы создали собственный шрифт или приобрели на него права, а теперь хотите опубликовать его в разных форматах, воспользуйтесь преобразователем на сайте FontSquirrel[1]. Он предоставляет как преобразованные шрифты, так и таблицу стилей с необходимым кодом @font-face. Обязательно убедитесь в том, что лицензия на шрифт разрешает такой вид использования.

Форматы шрифтов

Шрифты существуют во многих форматах. Выбор формата, который должен поставляться посетителям сайта, зависит от браузера.

Форматы и их поддержка браузерами:

- Embedded OpenType (EOT) [IE5–8]
- TrueType (TTF) [IE9, F3.5, C4, S4]
- OpenType (OTF) [IE9, F3.5, C4, S4, O10.5]
- Scalable Vector Graphics (SVG) [IOS]
- Web Open Font (WOFF) [IE9, F3.6]

Браузеры Internet Explorer до версии 9 поддерживали только формат, называемый EOT (Embedded OpenType). Другие браузеры также поддерживают более распространенные форматы TrueType и OpenType.

Microsoft, Opera и Mozilla совместно создали формат Web Open Font Format, поддерживающий сжатие без потерь и расширенные возможности лицензирования для создателей шрифтов.

Чтобы ваше решение работало во всех браузерах, шрифты придется публиковать в нескольких форматах.

Смена шрифта

Нужный нам шрифт доступен на сайте FontSquirrel[2] в форматах TrueType, WOFF, SVG и EOT — как раз то, что нужно.

Использование шрифта состоит из двух шагов: определения шрифта и присоединения его к элементам. Включите в таблицу стилей блога следующий код:

---

[1] http://www.fontsquirrel.com/fontface/generator

[2] Вы можете загрузить его с сайта http://www.fontsquirrel.com/fonts/Garogier или взять из примеров кода книги.

css3fonts/style.css

```
@font-face {
  font-family: 'GarogierRegular';
  src: url('fonts/Garogier_unhinted-webfont.eot');
  src: url('fonts/Garogier_unhinted-webfont.woff')
format('woff'),
       url('fonts/Garogier_unhinted-webfont.ttf')
format('truetype'),
       url('fonts/Garogier_unhinted-webfont.
svg#webfontew0qE0O9') format('svg');
font-weight: normal;
}
```

Сначала мы определяем семейство шрифтов по имени, а затем указываем источники. Версия в формате EOT ставится на первое место, чтобы браузер IE увидел ее немедленно, после чего перечисляются другие источники. Браузер пользователя перебирает их, пока не найдет подходящий.

Теперь определяемое семейство шрифтов можно использовать в таблице стилей. Мы изменяем исходный стиль шрифта и придаем ему следующий вид.

css3fonts/style.css

```
body{
  font-family: "GarogierRegular";
}
```

После этого простого изменения текст страницы выводится новым шрифтом, как показано на рис. 8.6.

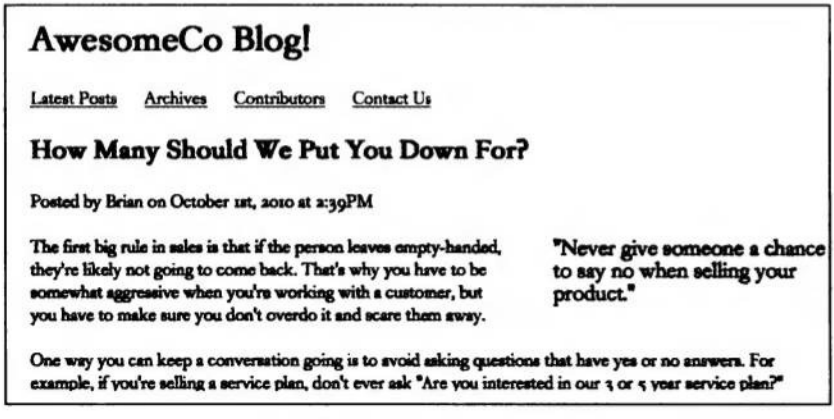


Рис. 8.6. Блог с новым шрифтом

В современных браузерах изменить шрифт относительно несложно, но мы также должны позаботиться о браузерах, в которых такая возможность пока не поддерживается.

Обходное решение

Обходные решения для разных версий IE и других браузеров уже были представлены ранее, однако также необходимо позаботиться о том, чтобы страницы нормально читались в браузерах без поддержки `@font-face`.

Мы предоставили альтернативные версии шрифта Garogier, но при его применении не были указаны никакие резервные шрифты. А это означает, что если браузер не поддерживает вывод шрифтом Garogier, он будет использовать шрифт по умолчанию. Такое решение не идеально.

Стек шрифтов (font stack) представляет собой список шрифтов, упорядоченных по приоритету. Сначала указывается шрифт, наиболее желательный для отображения, а затем все остальные шрифты в порядке убывания предпочтения.

Создавая стек шрифтов, не жалейте времени на поиск оптимальных резервных вариантов. Шрифты должны быть близки друг к другу по межбуквенным интервалам, толщине линий и внешнему виду в целом. Превосходная статья на эту тему опубликована на сайте UnitInteractive[1].

Давайте заменим наше определение шрифта следующим:

`css3fonts/style.css`

```
font-family: "GarogierRegular", Georgia,
             "Palatino", "Palatino Linotype",
             "Times", "Times New Roman", serif;
```

Обширный набор резервных шрифтов поможет сохранить внешний вид шрифта. Возможно, наша подборка не идеальна, но в любом случае она лучше шрифта по умолчанию, который иногда очень плохо читается.

Шрифты — один из важнейших факторов, определяющих внешнюю привлекательность и удобочитаемость страницы. Поэкспериментируйте с результатами своей работы. К вашим услугам множество шрифтов, бесплатных и коммерческих.

---

[1] http://unitinteractive.com/blog/2008/06/26/better-css-font-stacks/

Перспективы

В этой главе были рассмотрены некоторые усовершенствования традиционных методов веб-разработки в CSS. Впрочем, мы затронули лишь малую часть этой обширной темы. В спецификации CSS3 упоминаются 3D-преобразования и даже простая анимация; это означает, что для реализации простейших интерфейсных эффектов вместо JavaScript можно использовать CSS3 (по аналогии с `:hover`).

Кроме того, некоторые браузеры уже поддерживают множественные фоновые изображения и градиентное обрамление. Также следите за усовершенствованиями в области страничного контента — такими, как переходящие области заголовков/завершителей и поддержка нумерации страниц.

И наконец, модули CSS3 (когда работа над ними будет завершена) значительно упростят создание полнофункциональных, более совершенных и внешне привлекательных интерфейсных элементов. Следите за новостями!

III

За пределами HTML5

9 Работа с данными на стороне клиента

До настоящего момента мы рассматривали разметку HTML5 и CSS3. В этой части книги мы обратимся к технологиям и функциональности, имеющем косвенное отношение к HTML5. Например, поддержка междокументной передачи информации и автономной работы позволяет организовать взаимодействие между доменами и создавать решения, которые могут использоваться в автономном (offline) режиме.

Некоторые технологии — такие, как Web Storage, Web SQL Databases и Web Sockets — появились на базе спецификации HTML5. Другие (например, Geolocation) никогда в эту спецификацию не входили, но создатели браузеров и разработчики связывают Geolocation с HTML5, потому что спецификация реализуется в них наряду с другими возможностями.

Именно такие технологии рассматриваются в этой части книги. При этом особое внимание уделяется тем возможностям, которые можно использовать уже сегодня. Также целая глава будет посвящена обсуждению дальнейшего прогнозируемого развития в этой области. Начнем с Web Storage и Web SQL Storage — двух спецификаций, позволяющих организовать хранение данных на стороне клиента.

Вам нравилось работать с cookie? Вот и мне не нравилось. С cookie было неудобно возиться со времен их появления, однако с этими неудобствами приходилось мириться, потому что другого способа хранения данных на клиентском компьютере не было. Чтобы использовать cookie, необходимо было присвоить ему имя и задать срок действия. Код JavaScript, в котором выполнялись эти операции, обычно «заворачивался» в функцию, чтобы разработчик мог просто пользоваться им, не задумываясь над тем, как этот код работает:

html5_localstorage/setcookie.js

```
// Из http://www.javascripter.net/faq/settinga.htm
function SetCookie(cookieName,cookieValue,nDays) {
  var today = new Date();
  var expire = new Date();
  if (nDays==null || nDays==0) nDays=1;
  expire.setTime(today.getTime() + 3600000*24*nDays);
  document.cookie = cookieName+"="+escape(cookieValue)
    + «;expires="+expire.toGMTString();
}
```

Кроме труднозапоминаемого синтаксиса, также приходится учитывать проблемы безопасности. Некоторые сайты используют cookie для отслеживания поведения пользователей в Интернете, поэтому пользователи тем или иным образом отключают cookie.

В HTML5 появились новые технологии хранения данных на стороне клиента: Web Storage (с использованием `localStorage` или `sessionStorage`)[1] и Web SQL Databases[2]. Они просты в использовании, обладают чрезвычайно широкими возможностями и в достаточной степени безопасны. И что самое лучшее, они уже реализованы в нескольких браузерах, включая Mobile Safari (iOS) и браузер Android 2.0. Тем не менее формально эти технологии не являются частью спецификации HTML5, — они были выделены в отдельные спецификации.

Хотя `localStorage`, `sessionStorage` и Web SQL Databases не могут заменить cookie, которые должны передаваться между клиентом и сервером (как в веб-инфраструктурах, использующих cookie для хранения состояния между запросами), они могут использоваться для хранения данных, представляющих интерес только для пользователя (например, визуальных настроек или предпочтений). Также они хорошо подходят для построения мобильных приложений, которые могут работать в браузере без подключения к Интернету. Многие веб-приложения в настоящее время для сохранения пользовательских данных обращаются к серверу, но с появлением новых механизмов хранения необходимость в подключении к Интернету отпала. Пользовательские данные можно хранить локально, создавая их резервные копии при необходимости.

Объединение этих технологий с новыми средствами автономной работы HTML5 открывает возможность построения полноценных приложений баз данных, работающих в браузере в широком спектре платформ,

---

[1] http://www.whatwg.org/specs/web-apps/2007-10-26/#storage

[2] http://www.whatwg.org/specs/web-apps/2007-10-26/#sql

от настольных компьютеров до iPad и телефонов на платформе Android. Вы научитесь использовать эти технологии для сохранения пользовательских настроек и создания простой базы данных.

В этой главе рассматриваются следующие возможности[1]:

`localStorage`

Хранение данных в виде пар «ключ/значение» с привязкой к домену и сохранением данных между сеансами. [C5, F3.5, S4, IE8, O10.5, IOS, A]

`sessionStorage`

Хранение данных в виде пар «ключ/значение» с привязкой к домену и уничтожением данных при завершении сеанса. [C5, F3.5, S4, IE8, O10.5, IOS, A]

Web SQL Databases

Полноценные реляционные базы данных с поддержкой создания таблиц, вставки, обновления, удаления, выборки и транзакций, с привязкой к домену и сохранением данных между сеансами. [C5, S3.2, O10.5, IOS3.2, A2]

Offline Web Applications

Кэширование файлов для автономного использования; позволяет приложениям работать без подключения к Интернету. [C4, S4, F3.5, O10.6, IOS3.2, A2]

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

Рецепт 20. Сохранение настроек с использованием localStorage

Механизм локального хранения `localStorage` предоставляет в распоряжение разработчика очень простой способ хранения данных на клиентском компьютере. По сути, он представляет собой хранилище для пар «имя/значение», встроенное в браузер.

Информация, хранимая в `localStorage`, сохраняется между сеансами и не может читаться другими сайтами, потому что доступ к ней ограничивается текущим посещаемым доменом[1].

Фирма AwesomeCo занимается разработкой нового портала для своих клиентов и хочет, чтобы посетители могли изменять размер текста, фон и цвет текста на сайте. Для хранения настроек будет использоваться механизм `localStorage`, чтобы сохраненные данные не терялись между сеансами. После завершения работы у нас должен получиться прототип, показанный на рис. 9.1.

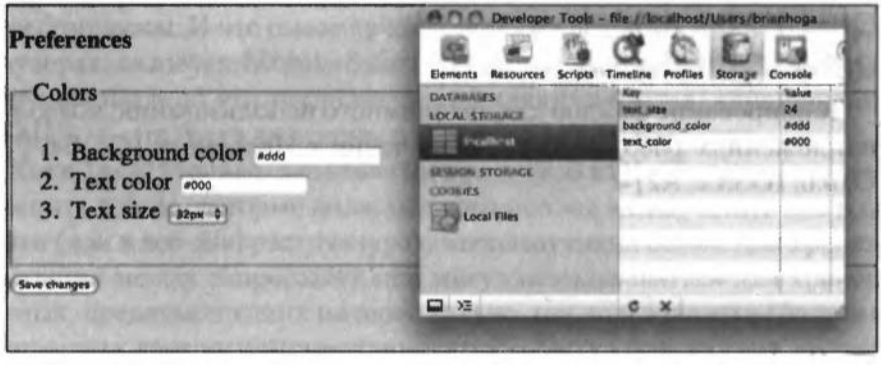


Рис. 9.1. Для локального хранения пользовательских настроек используется механизм localStorage

Построение формы

При построении формы ввода настроек будет использоваться семантическая разметка HTML5 и новые элементы, описанные в главе 3. Пользователю предоставляется возможность выбрать цвет текста, цвет фона, а также размер шрифта.

---

[1] Будьте внимательны при локальной разработке. Например, если вы работаете на localhost, переменные могут перепутаться!

html5_localstorage/index.html

```
<p><strong>Preferences</strong></p>
<form id="preferences" action="save_prefs"
      method="post" accept-charset="utf-8">
  <fieldset id="colors" class="">
    <legend>Colors</legend>
    <ol>
      <li>
        <label for="background_color">Background color</label>
        <input type="color" name="background_color"
               value="" id="background_color">
      </li>
      <li>
        <label for="text_color">Text color</label>
        <input type="color" name="text_color"
               value="" id="text_color">
      </li>
      <li>
        <label for="text_size">Text size</label>
        <select name="text_size" id="text_size">
          <option value="16">16px</option>
          <option value="20">20px</option>
          <option value="24">24px</option>
          <option value="32">32px</option>
        </select>
    </ol>

  </fieldset>

  <input type="submit" value="Save changes">
</form>
```

Для простоты цвета задаются шестнадцатеричными кодами HTML.

Сохранение и загрузка настроек

Для работы с системой `localStorage` достаточно обратиться к объекту `window.localStorage()` из кода JavaScript. Присваивание пары «имя/значение» выполняется очень просто.

html5_localstorage/index.html

```
localStorage.setItem("background_color",
$("#background_color").val());
```

Не сложнее выполняется и чтение сохраненных данных.

html5_localstorage/index.html

```
var bgcolor = localStorage.getItem("background_color");
```

Напишем вспомогательный метод для сохранения всех настроек с формы.

html5_localstorage/index.html

```
function save_settings(){
  localStorage.setItem("background_color",
  $("#background_color").val());
  localStorage.setItem("text_color", $("#text_color").val());
  localStorage.setItem("text_size", $("#text_size").val());
  apply_preferences_to_page();
}
```

Аналогичный метод загружает данные из системы `localStorage` и заполняет ими поля формы.

```
function load_settings(){
  var bgcolor = localStorage.getItem("background_color");
  var text_color = localStorage.getItem("text_color");
  var text_size = localStorage.getItem("text_size");

  $("#background_color").val(bgcolor);
  $("#text_color").val(text_color);
  $("#text_size").val(text_size);

  apply_preferences_to_page();
}
```

Этот метод также вызывает другой метод `apply_preferences_to_page()`, заполняющий загруженными данными элементы страницы. Он описывается в следующем рецепте.

Применение настроек

Теперь, когда мы можем прочитать настройки из `localStorage`, загруженные данные необходимо применить к странице. Все настройки в нашем примере связаны с CSS, а для изменения стилей любого элемента будет использоваться библиотека jQuery.

html5_localstorage/index.html

```
function apply_preferences_to_page(){
  $("body").css("backgroundColor", $("#background_color").
val());
  $("body").css("color", $("#text_color").val());
  $("body").css("fontSize", $("#text_size").val() + "px");
}
```

Остается активизировать всю эту логику при готовности документа.

html5_localstorage/index.html

```
$(function(){

  load_settings();

  $('form#preferences').submit(function(event){
    event.preventDefault();
    save_settings();
  });
});
```

Обходное решение

Метод `localStorage` работает только в последней версии Internet Explorer, Firefox, Chrome и Safari, поэтому нам понадобится обходное решение для более старых браузеров. Есть два способа: хранить информацию на сервере или на стороне клиента в cookie.

Хранение на стороне сервера

Если в вашей системе используются учетные записи пользователей, рассмотрите возможность сохранения настроек в профиле пользователя в вашем приложении. Когда пользователь входит в систему, вы проверяете, существуют ли настройки на стороне клиента, и если не существуют, — загружаете их с сервера. В этом случае настройки пользователей будут успешно загружаться в любом браузере и на любом компьютере.

Чтобы данные хранились на сервере, обеспечьте отправку данных формой на сервер — не запрещайте отправку по умолчанию в коде JavaScript.

Если пользователь отключил JavaScript, то хранение данных на стороне сервера становится единственным работоспособным решением, так

как приложение можно написать таким образом, чтобы оно загружало настройки из базы данных, а не из хеша `localStorage`. Кроме того, этот метод остается единственно возможным и в том случае, если сохраняется более 4 Кбайт данных (максимальный объем данных, хранимых в cookie).

Cookie и JavaScript

Другое обходное решение основано на проверенной комбинации «cookie/JavaScript». Используя хорошо известный сценарий для работы с cookie с сайта Quirksmode[1], мы построим собственное альтернативное решение для `localStorage`.

Проверить поддержку `localStorage` браузером несложно. Для этого достаточно проверить существование метода `localStorage` объекта `window`.

html5_localstorage/index.html

```
if (!window.localStorage){
}
```

Также понадобятся методы записи cookie, которые мы позаимствуем из статьи на сайте Quirksmode. Включите следующие функции JavaScript в сценарный блок:

html5_localstorage/index.html

```
function createCookie(name,value,days) {
  if (days) {
    var date = new Date();
    date.setTime(date.getTime()+(days*24*60*60*1000));
    var expires = "; expires="+date.toGMTString();
  }
  else var expires = "";
  document.cookie = name+"="+value+expires+"; path=/";
}

function readCookie(name) {
  var result = ""
  var nameEQ = name + "=";
  var ca = document.cookie.split(';');
```

[1] http://www.quirksmode.org/js/cookies.html

```
  for(var i=0;i < ca.length;i++) {
    var c = ca[i];
    while (c.charAt(0)==' ') c = c.substring(1,c.length);
    if (c.indexOf(nameEQ) == 0){
      result = c.substring(nameEQ.length,c.length);
    }else{
      result = "";
    }
  }
  return(result);
}
```

Наконец, мы создаем объект `localStorage`, использующий cookie в своей внутренней реализации. Простейшая реализация может выглядеть так.

html5_localstorage/index.html

```
 1  localStorage = (function () {
-    return {
-      setItem: function (key, value) {
-        createCookie(key, value, 3000)
 5      },
-
-      getItem: function (key) {
-        return(readCookie(key));
-      }
10   };
-  })();
```

SESSIONSTORAGE

Механизм localStorage хорошо подходит для хранения данных, которые должны сохраняться даже после закрытия браузера пользователем. Но иногда требуется хранить информацию только пока браузер остается открытым и удалять ее при завершении сеанса. Здесь на помощь приходит механизм sessionStorage. По основным принципам работы он очень похож на localStorage, но содержимое sessionStorage стирается при завершении браузерного сеанса. Для работы с этой системой хранения вместо объекта localStorage следует обратиться к объекту sessionStorage:

```
sessionStorage.setItem('name', 'Brian
Hogan');
var name = sessionStorage.getItem('name');
```

Обходное решения для sessionStorage сводится к простому удалению cookie при закрытии браузера.

Обратите внимание на строку 4: в ней создается cookie со сроком действия 3000 дней. Создать cookie с неограниченным сроком действия невозможно, поэтому мы выбираем невообразимо долгий срок.

Извне наша базовая реализация `localStorage` выглядит практически так же. Если вам потребуется удалять данные или стереть все содержимое хранилища, придется проявить побольше изобретательности. В идеале в ближайшем будущем обходное решение можно будет удалить, оставив только браузерные операции `localStorage()`.

Рецепт 21. Хранение информации в реляционной базе данных на стороне клиента

Механизмы `localStorage` и `sessionStorage` предоставляют простые средства хранения пар «имя/значение» на клиентском компьютере, но иногда требуется нечто большее. Возможность хранения информации в реляционных базах данных впервые появилась в спецификации HTML5. Позднее она была выделена в отдельную спецификацию, которая называется Web SQL Storage[1]. Каждый, кто хотя бы в общих чертах понимает, как пишутся команды SQL, практически сразу же сможет пользоваться новыми возможностями. Чтобы вы лучше поняли, как это делается, мы воспользуемся средствами Web SQL Storage для создания, выборки, обновления и удаления заметок, хранящихся в базе данных на стороне клиента.

Операции с базой данных в браузере

Спецификация и ее реализации позволяют нам выполнять операции вставки, выборки, обновления и удаления записей.

Фирма AwesomeCo хочет разработать для своей группы сбыта простое приложение для ведения заметок во время деловых поездок. Пользователь должен иметь возможность как создавать новые заметки, так и обновлять и удалять уже существующие. Для внесений изменений в существующие заметки необходимо прочитать их из базы данных.

Для выполнения этих операций необходимо реализовать следующие команды SQL.

Операция	Команда
Создание заметки	`INSERT INTO notes (title, note) VALUES("Test", "This is a note");`
Чтение всех заметок	`SELECT id, title, note FROM notes;`
Чтение конкретной заметки	`SELECT id, title, note FROM notes where id = 1;`
Обновление заметки	`UPDATE notes set title = "bar", note = "Changed" where id = 1;`
Удаление заметки	`DELETE FROM notes where id = 1;`

[1] http://dev.w3.org/html5/webdatabase/

ВОПРОС/ОТВЕТ

Разве спецификация Web SQL Database еще жива?

В ноябре 2010 года рабочая группа, занимавшаяся сопровождением спецификации, объявила о том, что она прекращает работу, и отныне все усилия будут направлены на спецификацию IndexedDB. Спецификация Web SQL Database обсуждается в книге только потому, что она уже реализована в браузерах на базе Webkit, включая все устройства iOS и Android, Safari и Google Chrome. В отличие от спецификации IndexedDB, нигде не реализованной на момент написания книги, вы можете использовать Web SQL Databases в своих проектах уже сейчас. Может оказаться, что ее возможности идеально подойдут для ваших проектов.

Интерфейс приложения

Интерфейс приложения для работы с заметками состоит из левой боковой панели со списком всех существующих заметок и формы с названием заметки и большим полем с текстом самой заметки. Примерный вид того, что мы создаем, представлен на рис. 9.2.

Начнем с разметки пользовательского интерфейса.

html5sql/index.html

```
<!doctype html>

<html>
  <head>
    <title>AwesomeNotes</title>
    <link rel="stylesheet" href="style.css">

  <script type="text/javascript"
          charset="utf-8"
          src=
          "http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/
          jquery.min.js">
  </script>

  <script type="text/javascript"
          charset="utf-8" src="javascripts/notes.js">
  </script>

  </head>

  <body>
```

```
    <section id="sidebar">
      <input type="button" id="new_button" value="New note">
      <ul id="notes">
      </ul>
    </section>

    <section id="main">
      <form>
        <ol>
          <li>
            <input type="submit" id="save_button" value="Save">
            <input type="submit" id="delete_button"
            value="Delete">
          </li>
          <li>
            <label for="title">Title</label>
            <input type="text" id="title">
          </li>
          <li>
            <label for="note">Note</label>
            <textarea id="note"></textarea>
          </li>
        </ol>
      </form>
    </section>

  </body>
</html>
```

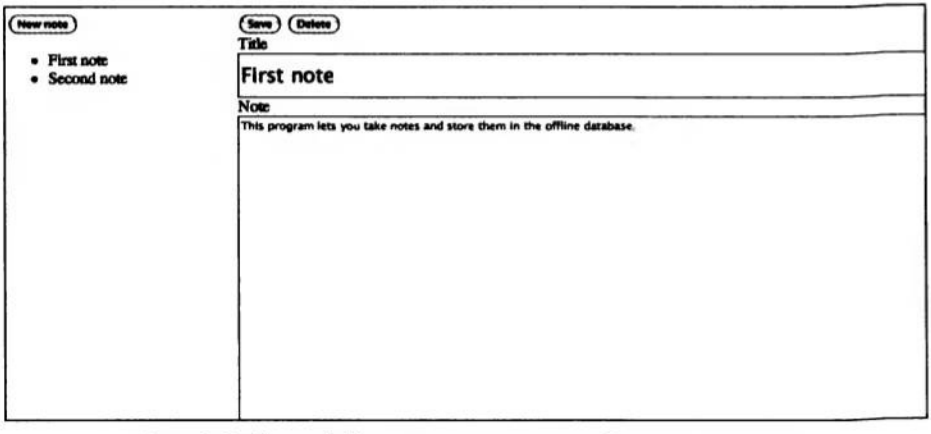


Рис. 9.2. Интерфейс приложения для работы с заметками

Мы определяем боковую панель и области основного контента в тегах `section`. Каждому важному элементу пользовательского интерфейса (такому, как кнопка `Save`) назначается идентификатор; он упростит поиск элементов для подключения обработчиков событий.

Чтобы наше приложение было больше похоже на рис. 9.2, мы применим к нему стилевое оформление. Файл *style.css* выглядит так.

html5sql/style.css

```
#sidebar, #main{
  display: block;
  float: left;
}

#sidebar{
  width: 25%;
}

#main{
  width: 75%;
}

form ol{
  list-style: none;
  margin: 0;
  padding: 0;
}
form li{
  padding: 0;
  margin: 0;
}

form li label{
  display:block;
}

#title, #note{
  width: 100%;
  font-size: 20px;
  border: 1px solid #000;
}

#title{
  height: 20px;
}
```

```
#note{
  height: 40px;
}
```

Таблица стилей отключает маркеры, задает размеры текстовых областей и формирует двухстолбцовый макет. Разобравшись с интерфейсом, можно переходить к написанию кода JavaScript для выполнения операций с базой данных.

Подключение к базе данных

Работа с базой данных начинается с создания подключения.

html5sql/javascripts/notes.js

```
// Ссылка на базу данных
var db = null;

// Создание подключения к локальной базе данных
connectToDB = function()
{
  db = window.openDatabase('awesome_notes', '1.0',
                           'AwesomeNotes Database',
1024*1024*3);
};
```

В начале сценария объявляется переменная db. Через эту переменную с базой данных будут работать остальные созданные нами методы[1]. Затем мы объявляем метод для подключения к базе данных, использующий метод `window.openDatabase`. При вызове передается имя базы данных, номер версии, описание и размер.

Создание таблицы

Таблица для хранения заметок состоит из трех столбцов.

Столбец	Описание
id	Уникальный идентификатор поля. Первичный ключ, целый тип, автоматическое увеличение
title	Название заметки
Note	Текст заметки

[1] Объявление переменной с глобальной видимостью — не лучшее решение. В нашем примере мы постарались сделать код JavaScript как можно более простым.

Метод создания таблицы выглядит так.

html5sql/javascripts/notes.js

```
createNotesTable = function()
{
  db.transaction(function(tx){
    tx.executeSql(
      "CREATE TABLE notes (id INTEGER \
       PRIMARY KEY, title TEXT, note TEXT)", [],
      function(){ alert('Notes database created successfully!');
},
      function(tx, error){ alert(error.message); } );
  });
};
```

Команда SQL выполняется в транзакции с двумя методами обратного вызова: для успешного выполнения и для ошибки. Эта схема будет использоваться для всех действий.

Обратите внимание: во втором параметре метода `executeSql()` также передается массив. Он предназначен для связывания подстановочных заполнителей в SQL с переменными. Это позволяет избежать конкатенации, а принцип действия напоминает подготовленные команды (prepared statements) в других языках. В нашем случае массив пуст, потому что запрос не содержит заполнителей.

Итак, таблица готова; теперь можно запрограммировать операции с ней.

Загрузка заметок

В процессе загрузки приложение должно подключиться к базе данных, создать таблицу (если она не существует), а затем прочитать из базы все существующие заметки.

html5sql/javascripts/notes.js

```
// Загрузка всех записей из таблицы notes базы данных
fetchNotes = function(){
  db.transaction(function(tx) {
    tx.executeSql('SELECT id, title, note FROM notes', [],
      function(SQLTransaction, data){
        for (var i = 0; i < data.rows.length; ++i) {
            var row = data.rows.item(i);
            var id = row['id'];
```

```
            var title = row['title'];

            addToNotesList(id, title);
        }
      });
  });
};
```

Метод читает результаты из базы данных. Если чтение проходит успешно, он перебирает записи и вызывает метод `addNoteToList`, который был определен следующим образом:

html5sql/javascripts/notes.js

```
// Включение нового элемента в список заметок с идентификатором
и названием
addToNotesList = function(id, title){
  var notes = $("#notes");
  var item = $("<li>");
  item.attr("data-id", id);
  item.html(title);
  notes.append(item);
};
```

Идентификатор записи сохраняется в пользовательском атрибуте данных. Он будет использован для поиска загружаемой записи, когда пользователь щелкает на элементе списка. Созданный элемент включается в неупорядоченный список нашего интерфейса с идентификатором `notes`. Теперь нужно добавить код загрузки этого элемента списка на форме при выделении заметки в списке.

Выборка конкретной записи

Можно было бы добавить событие `click` для каждого элемента списка, но более эффективное решение заключается в отслеживании всех щелчков на неупорядоченном списке и последующем определении выбранного элемента списка. В этом случае при добавлении нового элемента в список (скажем, при добавлении новой заметки) нам не придется добавлять новое событие `click`.

В функцию jQuery включается следующий код:

html5sql/javascripts/notes.js

```
$("#notes").click(function(event){
  if ($(event.target).is('li')) {
```

```
    var element = $(event.target);
    loadNote(element.attr("data-id"));
  }

});
```

В нем вызывается метод `loadNote`, который выглядит так.

html5sql/javascripts/notes.js

```
loadNote = function(id){
  db.transaction(function(tx) {
    tx.executeSql('SELECT id, title,
    note FROM notes where id = ?', [id],
      function(SQLTransaction, data){
        var row = data.rows.item(0);
        var title = $("#title");
        var note = $("#note");

        title.val(row["title"]);
        title.attr("data-id", row["id"]);
        note.val(row["note"]);
        $("#delete_button").show();

      });
  });
}
```

Этот метод имеет много общего с приведенным ранее методом `fetchNotes()`. Он выполняет команду SQL, после чего обрабатывается логика успешного пути выполнения. На этот раз команда содержит заполнитель (вопросительный знак), а подставляемое значение передается во втором параметре-массиве.

Найденная запись отображается на форме. Метод также активизирует кнопку `Delete` и встраивает идентификатор записи в пользовательский атрибут данных для удобства обновления. Кнопка `Save` проверяет существование записи с таким идентификатором. Если запись существует, она обновляется. Если запись не существует, мы считаем, что создается новая запись. Давайте реализуем эту часть логики.

Вставка, обновление и удаление записей

Когда пользователь щелкает на кнопке `Save`, должен срабатывать код вставки новой или обновления существующей записи. Чтобы добавить

к кнопке Save обработчик события click, мы включаем следующий фрагмент в функцию jQuery:

html5sql/javascripts/notes.js

```
$("#save_button").click(function(event){
  event.preventDefault();
  var title = $("#title");
  var note = $("#note");

  if(title.attr("data-id")){
    updateNote(title, note);
  }else{
    insertNote(title, note);
  }
});
```

Метод проверяет атрибут data-id поля title формы. Если идентификатор отсутствует, форма считает, что мы вставляем новую запись, и вызывает метод insertNote.

html5sql/javascripts/notes.js

```
insertNote = function(title, note)
{
  db.transaction(function(tx){
    tx.executeSql("INSERT INTO notes (title, note) VALUES (?,
?)",
                  [title.val(), note.val()],
      function(tx, result){
        var id = result.insertId ;
        alert('Record ' + id+ ' saved!');
        title.attr("data-id", result.insertId );
        addToNotesList(id, title.val());
        $("#delete_button").show();

      },
      function(){
        alert('The note could not be saved.');
      }
    );
  });
};
```

Метод insertNote() вставляет запись в базу данных и использует свойство insertId набора для получения идентификатора только

что вставленной записи. Идентификатор связывается с полем формы `title` как пользовательский атрибут данных, после чего вызов метода `addToNotesList()` включает заметку в список на боковой панели.

На следующем шаге выполняется обработка обновлений. Метод `updateNote()` очень похож на остальные методы, добавленные ранее.

html5sql/javascripts/notes.js

```
updateNote = function(title, note)
{
  var id = title.attr("data-id");
  db.transaction(function(tx){
    tx.executeSql("UPDATE notes set title = ?,
    note = ? where id = ?",
                  [title.val(), note.val(), id],
      function(tx, result){
        alert('Record ' + id + ' updated!');
        $("#notes>li[data-id=" + id + "]").html(title.val());
      },
      function(){
        alert('The note was not updated!');
      }
    );
  });
};
```

Если команда обновления была выполнена успешно, мы обновляем заголовок заметки в списке; элемент ищется по полю `data-id` со значением только что обновленного идентификатора.

Удаление записей происходит практически так же. Нам понадобится обработчик для события `delete`.

html5sql/javascripts/notes.js

```
$("#delete_button").click(function(event){
  event.preventDefault();
  var title = $("#title");
  deleteNote(title);
});
```

Сам метод `delete` не только удаляет запись из базы данных, но и исключает ее из списка заметок на боковой панели.

html5sql/javascripts/notes.js

```
deleteNote = function(title)
{
  var id = title.attr("data-id");
  db.transaction(function(tx){
    tx.executeSql("DELETE from notes where id = ?", [id],
      function(tx, result){
        alert('Record ' + id + ' deleted!');
        $("#notes>li[data-id=" + id + "]").remove();
      },
      function(){
        alert('The note was not deleted!');
      }
    );
  });
};
```

Остается очистить форму, чтобы создание новой записи не привело к случайному дублированию существующей.

Все вместе

Приложение почти готово. Осталось активизировать кнопку **New**, щелчок на которой очищает форму, чтобы пользователь мог создать новую запись после редактирования существующей. Для этого мы воспользуемся уже знакомой схемой — начнем с обработчика событий в функции jQuery для кнопки **New**.

html5sql/javascripts/notes.js

```
$("#new_button").click(function(event){
  event.preventDefault();
  newNote();
});
//end:newbutton

newNote();

});
```

Далее из поля `title` удаляется атрибут `data-id`, а в полях формы стираются текущие значения. Также в интерфейсе скрывается кнопка `Delete`.

html5sql/javascripts/notes.js

```
newNote = function(){
  $(«#delete_button").hide();
  var title = $("#title");
  title.removeAttr("data-id");
  title.val("");
  var note = $("#note");
  note.val("");
}
```

Новый метод `newForm` должен вызываться из функции jQuery при загрузке страницы, чтобы форма была готова к использованию. При этом кнопка `Delete` тоже останется скрытой.

Вот и все! Наше приложение работает на iPhone, устройствах Android и настольных компьютерах с Chrome, Safari и Opera. Однако, скорее всего, оно не будет работать в Firefox, и в Internet Explorer оно тоже не поддерживается.

Обходное решение

В отличие от других решений, хороших библиотек, которые бы позволяли нам реализовать функциональность SQL самостоятельно, не существует, поэтому реализовать приложение встроенными средствами Internet Explorer не удастся. Но если такое приложение, на ваш взгляд, будет полезно вашим пользователями, убедите их использовать браузер Google Chrome, работающий на всех платформах, для этого конкретного приложения. В такой практике нет ничего необычного, особенно когда переход на альтернативный браузер позволяет построить внутреннее приложение, которое будет работать также и на мобильных устройствах.

Другая альтернатива — использование плагина Google Chrome Frame[1]. Включите его в начало страницы HTML, прямо под тегом `head`.

html5sql/index.html

```
<meta http-equiv="X-UA-Compatible" content="chrome=1">
```

Этот фрагмент читается плагином Google Chrome Frame и активизирует его для данной страницы.

Если вы хотите проверить присутствие плагина и предложить вашим пользователям установить его, если плагин не найден, включите следующий фрагмент прямо над закрывающим тегом `body`:

[1] http://code.google.com/chrome/chromeframe/

html5sql/index.html

```
<script type="text/javascript"
  src=
  "http://ajax.googleapis.com/ajax/libs/chrome-frame/1/
  CFInstall.min.js">
</script>

<script>

  window.attachEvent("onload", function() {
    CFInstall.check({
      mode: "inline", // используется по умолчанию
      node: "prompt"
    });
  });
</script>
```

Тем самым вы предоставите пользователю возможность установки плагина, чтобы он мог работать с вашим сайтом.

Решение с Google Chrome Frame может оказаться неподходящим для веб-приложений, ориентированных на широкую публику, но оно достаточно хорошо подходит для внутренних приложений (вроде только что написанного нами). В некоторых корпоративных средах существуют IT-политики, запрещающие подобные решения; попробуйте добиться соответствующего разрешения, если вы окажетесь в такой ситуации. Установка плагина безусловно обойдется дешевле, чем разработка собственной СУБД на базе SQL.

Рецепт 22. Автономная работа

Поддержка автономной работы в HTML5[1] позволяет нам использовать HTML и сопутствующие технологии для построения приложений, работающих даже при отсутствии подключения к Интернету. Данная возможность особенно полезна при разработке приложений для мобильных устройств, на которых чаще происходит потеря подключения.

Технология работает в Firefox, Chrome и Safari, а также на устройствах iOS и Android 2.0. Тем не менее не существует обходного решения, которое бы реализовало поддержку автономной работы в Internet Explorer.

Фирма AwesomeCo только что закупила планшеты iPad для своей группы сбыта. Вам поручено сделать так, чтобы приложение, разработанное нами в предыдущем разделе, работало в автономном режиме. Благодаря файлу манифеста HTML5 сделать это будет несложно.

Определение кэша в манифесте

Файл манифеста содержит список всех файлов клиентской стороны веб-приложения, которые должны храниться в кэше для автономной работы. *Все* файлы, к которым будет обращаться приложение, должны быть перечислены в манифесте. Единственное исключение составляет файл с манифестом; он кэшируется автоматически.

Создайте файл с именем *notes.manifest*. Его содержимое должно выглядеть так.

html5offline/notes.manifest

```
CACHE MANIFEST
# v = 1.0.0
/style.css
/javascripts/notes.js
/javascripts/jquery.min.js
```

Изменение комментария с номером версии сообщает браузеру о необходимости загрузки новых версий файлов. Если вы изменяете код приложения, не забудьте обновить манифест.

Чтобы приложение работало автономно, вариант с размещением jQuery в Google уже не подходит. Следовательно, мы должны загрузить

[1] http://www.w3.org/TR/html5/offline.html

jQuery и изменить тег `script`, чтобы библиотека jQuery загружалась из папки *javascripts*.

`html5offline/index.html`

```
<script type="text/javascript"
  charset="utf-8"
  src="javascripts/jquery.min.js">
</script>
```

Файл манифеста необходимо связать с документом HTML. Для этого элемент `html` приводится к следующему виду:

`html5offline/index.html`

```
<html manifest="notes.manifest">
```

Вот и все! Впрочем, есть одна загвоздка — файл манифеста должен предоставляться веб-сервером, потому что манифест должен поставляться с типом MIME `text/cache-manifest`. Если вы используете Apache, укажите тип MIME в файле *.htaccess*.

`html5offline/.htaccess`

```
AddType text/cache-manifest .manifest
```

При первом запросе приложения `notes` файлы, перечисленные в манифесте, загружаются и кэшируются браузером. После этого можно отключиться от сети и использовать приложение столько раз, сколько потребуется.

Обязательно ознакомьтесь со спецификацией. Файл манифеста поддерживает много сложных настроек, которые вы можете использовать в своей работе. Например, можно указать, что некоторые файлы не должны кэшироваться и приложение никогда не должно обращаться к ним в автономном режиме; эта возможность быть полезна для предотвращения кэширования некоторых динамических файлов.

Манифест и кэширование

Пока вы работаете со своим приложением в режиме разработки, кэширование на веб-сервере следует отключить. По умолчанию многие веб-серверы кэшируют файлы, создавая заголовки, которые приказывают браузерам не загружать новую копию файла в течение заданного времени. Это может создать проблемы при добавлении новых записей в файл манифеста.

Если вы используете Apache, для запрета кэширования включите следующую запись в файл *.htaccess*:

html5offline/.htaccess

```
ExpiresActive On
ExpiresDefault "access"
```

Кэширование отключается для всего каталога, и для рабочего кода такие меры нежелательны. Однако они гарантируют, что браузер всегда будет запрашивать новую версию файла манифеста.

Если вы изменяете файл, указанный в манифесте, также отредактируйте файл манифеста — измените комментарий с номером версии.

Перспективы

Такие технологии, как `localStorage` и Web SQL Databases, позволяют разработчикам строить браузерные приложения, не нуждающиеся в подключении к веб-серверу. Такие приложения также работают на iPad и устройствах Android, а в сочетании с манифестами HTML5 появляется возможность строить полнофункциональные автономные приложения с использованием знакомых инструментов вместо специализированных, закрытых платформ. По мере того как эти возможности будут поддерживаться большим количеством браузеров, разработчики смогут шире использовать их; созданные ими приложения будут работать на многих платформах и устройствах. Данные будут сохраняться локально, с возможностью синхронизации при подключении.

Будущее технологии Web SQL Storage остается неясным. Mozilla не собирается реализовывать ее в Firefox, а W3C планирует переключиться на реализацию спецификации IndexedDB. Эта спецификация более подробно рассматривается в разделе 11.5, с. 244. Тем не менее технология Web SQL Storage поддерживается в устройствах iOS и Android в течение некоторого времени, и скорее всего, еще будет поддерживаться. Спецификация может быть исключительно полезной, если вы занимаетесь разработкой приложений в этой нише.

10 Взаимодействие с другими API

Многие интересные API, существование которых началось со спецификации HTML5, со временем преобразовались в отдельные проекты. Другие настолько тесно связаны с HTML5, что разработчикам (а иногда даже авторам) становится трудно различить их. В этой главе мы поговорим о таких API. Начнем с HTML5 History API для работы с историей просмотра, а затем перейдем к взаимодействию страниц на разных серверах с использованием Cross-Document Messaging API. Далее будут рассмотрены Web Sockets и Geolocation — два чрезвычайно мощных API, предназначенных для расширения интерактивности приложений.

Для построения приложений будут использоваться следующие API[1]:

History

Управление историей просмотра. [C5, S4, IE8, F3, O10.1 IOS3.2, A2]

Cross-Document Messaging

Передача сообщений между окнами с контентом, загруженным в разных доменах. [C5, S5, F4, IOS4.1, A2]

---

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

Web Sockets

Создание подключения с поддержкой состояния между браузером и сервером. [C5, S5, F4, IOS4.2]

Geolocation

Получение информации о широте и долготе. [C5, S5, F3.5, O10.6, IOS3.2, A2]

Рецепт 23. История просмотра

В спецификации HTML5 представлен API для управления историей просмотра[1]. В рецепте 12, «Создание обновляемых областей с улучшенной доступностью» (с. 107) мы построили прототип новой домашней страницы фирмы AwesomeCo с переключением основного контента при щелчке на одной из навигационных вкладок. У такого решения есть один недостаток: оно не поддерживает кнопку **Back** в браузере. Проблему можно решить при помощи некоторых трюков, но History API позволяет решить ее раз и навсегда.

Поддержка этого API проверяется следующим образом:

html5history/javascripts/application.js

```
function supportsHistory(){
   return !!(window.history && window.history.pushState);
}
```

Мы будем использовать этот метод каждый раз, когда нам потребуется работать с объектами History.

Хранение текущего состояния

Каждый раз, когда посетитель вызывает новую веб-страницу, браузер включает эту страницу в историю просмотра. Но когда пользователь вызывает новую вкладку, нам придется добавлять ее в историю просмотра вручную.

html5history/javascripts/application.js

```
 1    $("nav ul").click(function(event){
-      target = $(event.target);
-      if(target.is("a")){
-        event.preventDefault();
 5       if ( $(target.attr("href")).hasClass("hidden") ){
-
-          if(supportsHistory()){
-            var tab = $(target).attr("href");
-            var stateObject = {tab: tab};
10          window.history.pushState(stateObject, tab);
-          };
-
```

[1] http://www.w3.org/TR/html5/history.html

```
-           $(".visible").removeClass("visible").
                          addClass("hidden").hide();
-           $(target.attr("href")).removeClass("hidden").
                           addClass("visible").show();
15        };
-       };
-     });
-
-  });
```

Мы получаем идентификатор видимого элемента и добавляем состояние просмотра в историю браузера. В первом параметре метода `pushstate()` передается объект, с которым мы будем взаимодействовать позднее. Он будет использоваться для сохранения идентификатора вкладки, которая должна отображаться при возвращении пользователя к этой точке. Например, если пользователь щелкает на вкладке `Services`, в объекте состояния сохраняется запись `#services`.

Во втором параметре передается заголовок, который будет использоваться для идентификации состояния в истории просмотра. Он не имеет ничего общего с элементом `title` страницы и предназначен исключительно для идентификации записи в истории просмотра. В качестве значения этого параметра снова будет использован идентификатор вкладки.

Чтение предыдущего состояния

Добавить новое состояние в историю просмотра недостаточно — нужно еще написать код обработки изменения состояния. Когда пользователь щелкает на кнопке `Back`, срабатывает событие `window.onpopstate()`. Мы используем этот обработчик для отображения вкладки, сохраненной в объекте состояния.

`html5history/javascripts/application.js`

```
if(supportsHistory()){
  window.onpopstate = function(event) {
    if(event.state){
      var tab = (event.state["tab"]);
      $(".visible")
        .removeClass("visible")
        .addClass("hidden")
        .hide();
      $(tab)
```

```
        .removeClass("hidden")
        .addClass("visible")
        .show();
    }
  };
};
```

Мы читаем имя вкладки, а затем используем jQuery для поиска скрываемого элемента по идентификатору. Код сокрытия и отображения вкладок дублирует соответствующий фрагмент исходного кода. От дублирования следует избавиться посредством рефакторинга.

История по умолчанию

При первой загрузке страницы состояние истории просмотра будет пустым, поэтому нам придется задать ее вручную. Это можно сделать в определении метода `window.onpopstate()`:

html5history/javascripts/application.js

```
if(supportsHistory()){
  window.history.pushState( {tab: "#welcome"}, '#welcome');
  window.onpopstate = function(event) {
    if(event.state){
      var tab = (event.state["tab"]);
      $(".visible")
        .removeClass("visible")
        .addClass("hidden")
        .hide();
      $(tab)
        .removeClass("hidden")
        .addClass("visible")
        .show();
    }
  };
};
```

Теперь при открытии страницы мы сможем перебирать открывавшиеся ранее вкладки, используя историю просмотра[1].

[1] В процессе тестирования вам придется постоянно закрывать браузер и очищать историю просмотра. Иногда это бывает весьма утомительно.

Обходное решение

Наше решение работает в Firefox 4 и Safari 4, а также в Chrome 5, но в Internet Explorer оно работать не будет. Такие решения, как плагин jQuery Address[1], предоставляют аналогичную функциональность, однако мы не будем заниматься его реализацией, потому что это не столько обходное решение, сколько полная замена с множеством дополнительных возможностей. Следите за поддержкой истории просмотра в браузерах, возможно, использование этого API во всех браузерах сделает ваши приложения намного более удобными для пользователя.

[1] http://www.asual.com/jquery/address/

Рецепт 24. Передача информации между доменами

Клиентским веб-приложениям всегда запрещалось прямое взаимодействие со сценариями других доменов. Это ограничение было установлено ради безопасности пользователей[1]. Существует много хитроумных обходных путей, включая использование посредников на стороне сервера и трюков с URL. Однако теперь появился более удобный способ.

В спецификации HTML5 представлен API Cross-Document Messaging, обеспечивающий возможность обмена информацией между сценариями, размещенными в разных доменах. Например, форма *http://support.awesomecompany.com* может отправлять данные другому окну или `iframe`, контент которого размещается по адресу *http://www.awesomecompany.com*. Именно это и нужно сделать в нашем текущем проекте.

На новом сайте поддержки AwesomeCo имеется форма для работы с контактными данными. Начальник службы поддержки хочет, чтобы рядом с формой выводился список всех контактов из службы поддержки с адресами электронной почты. Эта информация будет поступать из системы управления контентом (CMS) или с другого сервера, поэтому мы можем встроить список контактов в `iframe`. Проблема в том, что начальник службы поддержки требует, чтобы при щелчке на имени в списке контактов соответствующий адрес электронной почты автоматически добавлялся на форму.

Задача решается относительно просто, но для качественного тестирования вашей конкретной конфигурации придется использовать веб-серверы. Примеры, которые мы будем рассматривать, не работают в некоторых браузерах без веб-сервера. За дополнительной информацией обращайтесь к приведенной ниже врезке «Простые веб-серверы».

Список контактов

Начнем с создания списка контактов. Базовая разметка выглядит так.

`html5xdomain/contactlist/public/index.html`

```
<ul id="contacts">
  <li>
    <h2>Sales</h2>
```

[1] Политика единого происхождения (Same Origin Policy) более подробно объясняется в статье https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript.

```
    <p class="name">James Norris</p>
    <p class="email">j.norris@awesomeco.com</p>
  </li>
  <li>
    <h2>Operations</h2>
    <p class="name">Tony Raymond</p>
    <p class="email">t.raymond@awesomeco.com</p>
  </li>
  <li>
    <h2>Accounts Payable</h2>
    <p class="name">Clark Greenwood</p>
    <p class="email">c.greenwood@awesomeco.com</p>
  </li>
  <li>
    <h2>Accounts Receivable</h2>
    <p class="name">Herbert Whitmore</p>
    <p class="email">h.whitmore@awesomeco.com</p>
  </li>
</ul>
```

ПРОСТЫЕ ВЕБ-СЕРВЕРЫ

Если вы не хотите возиться с настройкой экземпляров Apache или собственных серверов, попробуйте использовать простые серверы на базе Ruby, включенные в файлы примеров этой книги. Инструкции по настройке Ruby для вашей системы приведены в файле RUBY_README.txt в архиве примеров.

Чтобы запустить серверы, зайдите в каталог html5xdomain/contactlist и запустите файл server.rb следующей командой:

```
ruby server.rb
```

Сервер запускается на порте 4567. Сделайте то же самое с файлом server.rb из каталога html5xdomain/supportpage; сервер запускается на порте 3000. Чтобы сменить порт любого из этих серверов, отредактируйте файл server.rb.

Страница также загружает библиотеку jQuery, файл *application.js* и простую таблицу стилей. Директивы включаются в раздел head.

html5xdomain/contactlist/public/index.html

```
<script type="text/javascript"
  charset="utf-8"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.
min.js">
</script>
<script type="text/javascript"
```

```
  src="javascripts/application.js">
</script>
<link rel="stylesheet" href="style.css" type="text/css"
media="screen">
```

Таблица стилей для списка контактов выглядит так.

html5xdomain/contactlist/public/style.css

```
ul{
  list-style: none;
}

ul h2, ul p{margin: 0;}
ul li{margin-bottom: 20px;}
```

Просто сделана пара настроек, с которыми список выглядит чуть более аккуратно.

Отправка сообщения

Когда пользователь щелкает на строке списка контактов, мы читаем адрес электронной почты из элемента списка и отправляем сообщение родительскому окну. Метод `postMessage()` получает два параметра: само сообщение и исходный адрес окна. Весь обработчик событий выглядит так.

html5xdomain/contactlist/public/javascripts/application.js

```
$(function(){
  $("#contacts li").click(function(event){
    var email = ($(this).find(".email").html());
    var origin = "http://192.168.1.244:3000/index.html";
      window.parent.postMessage(email, origin);
  });
});
```

Если вы воспроизводите код примеров на своем компьютере, не забудьте изменить второй параметр — в нем должен передаваться фактический URL-адрес родительского окна[1].

[1] Это не совсем точно — в параметре может быть указан домен и даже шаблон. Но для нашего обходного решения параметр должен содержать полный URL-адрес, к тому же это требование улучшает безопасность системы.

Теперь нужно реализовать страницу, которая будет содержать этот фрейм и получать его сообщения.

Сайт поддержки

Сайт поддержки имеет очень похожую структуру, но для логической изоляции двух структур мы будем работать в другой папке (особенно если учесть, что сайт должен размещаться на другом веб-сервере). Также не забудьте включить ссылки на таблицу стилей, jQuery и новый файл *application.js*.

Страница поддержки содержит форму отправки сообщения и `iframe` со списком контактов.

`html5xdomain/supportpage/public/index.html`

```
<div id="form">
  <form id="supportform">
    <fieldset>
      <ol>
        <li>
          <label for="to">To</label>
          <input type="email" name="to" id="to">
        </li>
        <li>
          <label for="from">From</label>
          <input type="text" name="from" id="from">
        </li>
        <li>
          <label for="message">Message</label>
          <textarea name="message" id="message"></textarea>
        </li>
      </ol>
      <input type="submit" value="Send!">
    </fieldset>
  </form>
</div>

<div id="contacts">
  <iframe src="http://192.168.1.244:4567/index.html"></iframe>
</div>
```

Стилевое оформление реализуется следующим кодом CSS, который мы включим в файл *style.css*:

html5xdomain/supportpage/public/style.css

```
#form{
  width: 400px;
  float: left;
}
#contacts{
  width: 200px;
  float: left;
}
#contacts iframe{
  border: none;
  height: 400px;
}

fieldset{
  width: 400px;
  border: none;
}

fieldset legend{
  background-color: #ddd;
  padding: 0 64px 0 2px;
}

fieldset>ol{
  list-style: none;
  padding: 0;
  margin: 2px;
}

fieldset>ol>li{
  margin: 0 0 9px 0;
  padding: 0;
}
/* Текстовые поля размещаются с новой строки */
fieldset input, fieldset textarea{
  display:block;
  width: 380px;
}
fieldset input[type=submit]{
  width: 390px;
}
```

```
fieldset textarea{
  height: 100px;
}
```

Форма и `iframe` располагаются рядом друг с другом, а форма приводится к виду, показанному на рис. 10.1.

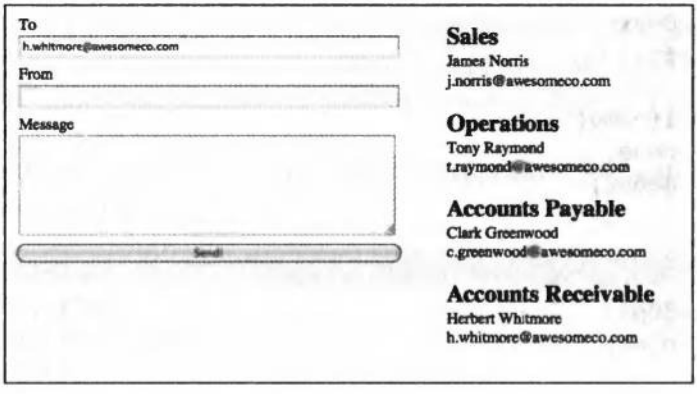


Рис. 10.1. Сайт поддержки

Получение сообщений

Событие `onmessage` инициируется при получении сообщения текущим окном. Сообщение передается в свойстве объекта `event`. Для регистрации события будет использоваться метод jQuery `bind()`, чтобы решение одинаково работало во всех браузерах.

`html5xdomain/supportpage/public/javascripts/application.js`

```
$(function(){
  $(window).bind("message",function(event){
    $("#to").val(event.originalEvent.data);
  });

});
```

Метод jQuery `bind()` инкапсулирует событие и блокирует доступ к некоторым из его свойств. Нужную информацию можно получить из свойства `originalEvent` объекта события.

Такое решение отлично работает в Firefox, Chrome, Safari или Internet Explorer 8. Теперь давайте заставим его работать в IE6 и 7.

Обходное решение

Для поддержки IE6 и 7 мы воспользуемся плагином jQuery Postback, эмулирующим междоменную передачу информации. Метод jQuery `getScript()` подгружает эту библиотеку только в том случае, если она действительно необходима (а для определения необходимости мы просто проверяем существование метода `postMessage()`).

Начнем с изменения списка контактов.

html5xdomain/contactlist/public/javascripts/application.js

```
if(window.postMessage){
  window.parent.postMessage(email, origin);
}else{
  $.getScript("javascripts/jquery.postmessage.js", function(){
    $.postMessage(email, origin, window.parent);
  });
}
```

Плагин jQuery Postmessage добавляет метод `postMessage()`, который почти не отличается от стандартного метода `postMessage()`.

Теперь обратимся к сайту поддержки. Мы воспользуемся тем же приемом с загрузкой библиотеки и вызовом добавленного метода `receiveMessage()`.

html5xdomain/contactlist/public/javascripts/application.js

```
if(window.postMessage){
  $(window).bind("message",function(event){
    $("#to").val(event.originalEvent.data);
  });
}else{
  $.getScript("javascripts/jquery.postmessage.js", function(){
    $.receiveMessage(
      function(event){
        $("#to").val(event.data);
      });
  });
}
```

Вот и все! Теперь наше приложение позволяет передавать информацию между окнами в полном наборе браузеров. Впрочем, это всего лишь начало; вы можете расширить эту методику для двустороннего обмена сообщениями. Любое окно может как отправлять, так и получать информацию; ознакомьтесь со спецификацией и подумайте, какие возможности она перед вами открывает!

Рецепт 25. Чат на базе Web Sockets

Веб-разработчики уже давно старались реализовать общение в реальном времени, но большинство реализаций базировалось на периодических обращениях к удаленному серверу из JavaScript для проверки изменений. Протокол HTTP не поддерживает состояния, поэтому браузер устанавливает связь с сервером, получает ответ и отключается. Выполнять содержательную работу в реальном времени через протокол, не поддерживающий состояния, достаточно трудно. В спецификации HTML5 представлена технология Web Sockets, которая позволяет браузеру создать подключение к удаленному серверу с поддержкой состояния[1]. На ее основе можно построить много интересных приложений. Чтобы понять, как работает технология Web Sockets, проще всего написать клиентское приложение для чата — что как раз и нужно фирме AwesomeCo.

AwesomeCo хочет реализовать на своем сайте поддержки простой чат на базе веб-технологий, при помощи которого персонал службы поддержки сможет общаться между собой (филиалы службы находятся в разных городах). Для реализации веб-интерфейса к серверу чата будет использована технология Web Sockets. Пользователи могут подключаться к серверу и отправлять ему сообщения, которые будут видны всем подключенным пользователям. Посетители смогут выбирать себе ники, отправляя сообщения вида «/nick brian» по аналогии с чат-протоколом IRC. К счастью, нам не придется писать чат-сервер самостоятельно, потому что эта работа уже была выполнена другим разработчиком[2].

Интерфейс чата

Мы создадим очень простой интерфейс, показанный на рис. 10.2, с формой для изменения ника пользователя, большой областью для вывода сообщений и, наконец, формой для отправки сообщений в чат.

На новой странице HTML5 добавляется разметка пользовательского интерфейса, состоящего из двух форм и тега `div` с сообщениями чата.

---

[1] Технология Web Sockets была выделена в отдельную спецификацию, которая находится по адресу http://www.w3.org/TR/websockets/.

[2] За дополнительной информацией о серверах обращайтесь к разделу «Серверы» (с. 226).

Рис. 10.2. Интерфейс чата

html5_websockets/public/index.html

```
<div id="chat_wrapper">
  <h2>AwesomeCo Help!</h2>
  <form id="nick_form" action="#" method="post" accept-
charset="utf-8">
    <p>
      <label>Nickname
        <input id="nickname" type="text" value="GuestUser"/>
      </label>
      <input type="submit" value="Change">
    </p>
  </form>

  <div id="chat">connecting....</div>

  <form id="chat_form" action="#" method="post" accept-
charset="utf-8">
    <p>
      <label>Message
        <input id="message" type="text" />
      </label>
      <input type="submit" value="Send">
    </p>
  </form>
</div>
```

Также необходимо добавить ссылки на таблицу стилей и файл JavaScript, содержащий код взаимодействия с сервером Web Sockets.

html5_websockets/public/index.html

```
<script src='chat.js' type='text/javascript'></script>
<link rel="stylesheet" href="style.css" media="screen">
```

Таблица стилей содержит следующие определения:

html5_websockets/public/style.css

```
 1  #chat_wrapper{
-    width: 320px;
-    height: 440px;
-    background-color: #ddd;
 5    padding: 10px;
-  }
-  #chat_wrapper h2{
-    margin: 0;
-  }
10
-  #chat{
-    width: 300px;
-    height: 300px;
-    overflow: auto;
15   background-color: #fff;
-    padding: 10px;
-  }
```

В строке 14 свойство `overflow` области сообщений задается таким образом, чтобы ее высота оставалась фиксированной, а не помещающийся текст скрывался (и мог просматриваться при помощи полос прокрутки).

После реализации интерфейса можно переходить к следующему шагу — написанию кода JavaScript, обеспечивающему взаимодействие с чат-сервером.

Взаимодействие с сервером

С каким бы сервером Web Socket мы ни работали, взаимодействие всегда строится по одной схеме. Мы создаем подключение к серверу, прослушиваем передаваемые сервером события и реагируем соответствующим образом.

Код файла *chat.js* начинается с создания подключения к серверу Web Sockets.

html5_websockets/public/chat.js

```
var webSocket = new WebSocket('ws://localhost:9394/');
```

Событие	Описание
onopen()	Успешное создание подключения к серверу
onmessage()	Отправка сообщения через подключение к серверу
onclose()	Потеря или закрытие подключения к серверу

Если подключение к серверу создано успешно, необходимо сообщить об этом пользователю. Метод `onopen()` определяется следующим образом:

html5_websockets/public/chat.js

```
webSocket.onopen = function(event){
  $('#chat').append('<br>Connected to the server');
};
```

Когда браузер открывает подключение к серверу, в окне чата выводится соответствующее сообщение. Далее необходимо выводить все сообщения, отправляемые чат-серверу. Для этого используется следующее определение метода `onmessage()`:

html5_websockets/public/chat.js

```
webSocket.onmessage = function(event){
  $('#chat').append("<br>" + event.data);
  $('#chat').animate({scrollTop: $('#chat').height()});
};
```

Сообщение возвращается в свойстве `data` объекта `event`; мы просто добавляем его в окно чата. В нашем примере перед каждым сообщением ставится перевод строки, чтобы сообщение выводилось в отдельной строке, но вы можете использовать любое другое оформление по вашему усмотрению.

Далее необходимо обеспечить обработку разрыва связи. Метод `onclose()` инициируется при закрытии подключения.

html5_websockets/public/chat.js

```
webSocket.onclose = function(event){
  $("#chat").append('<br>Connection closed');
};
```

Осталось связать текстовую область с формой чата, чтобы мы могли отправлять свои сообщения на сервер.

html5_websockets/public/chat.js

```
$(function(){
  $("form#chat_form").submit(function(e){
    e.preventDefault();
    var textfield = $("#message");
    webSocket.send(textfield.val());
    textfield.val("");
  });
})
```

Мы перехватываем событие формы `submit`, получаем значение поля формы и отправляем его чат-серверу методом `send()`.

Смена ника реализуется аналогичным образом, разве что перед отправляемым сообщением вставляется префикс «/nick». Чат-сервер распознает его и изменяет имя пользователя.

html5_websockets/public/chat.js

```
$("form#nick_form").submit(function(e){
  e.preventDefault();
  var textfield = $("#nickname");
  webSocket.send("/nick " + textfield.val());
});
```

Вот и все! Пользователи Safari 5 и Chrome 5 могут немедленно вступать в общение, проходящее в реальном времени. Конечно, мы еще должны позаботиться о пользователях браузеров, не имеющих встроенной поддержки Web Sockets. Обходное решение будет строиться на базе технологии Flash.

Обходное решение

Даже если браузер не поддерживает подключение через сокеты, в Adobe Flash такая поддержка реализована уже давно. Технология Flash обеспечит необходимое взаимодействие через сокеты, а благодаря библиотеке web-socket-js[1] обходное решение на базе Flash реализуется проще простого.

Загрузите копию плагина[2] и включите ее в свой проект. Далее в страницу необходимо включить три файла JavaScript.

---

[1] http://github.com/gimite/web-socket-js/

[2] http://github.com/gimite/web-socket-js/archives/master

html5_websockets/public/index.html

```
  <script type="text/javascript" src="websocket_js/swfobject.
  js"></script>
  <script type="text/javascript" src="websocket_js/FABridge.
  js"></script>
  <script type="text/javascript" src="websocket_js/web_socket.
  js"></script>

  <script src='chat.js' type='text/javascript'></script>
  <link rel="stylesheet" href="style.css" media="screen">

</head>
<body>
<div id="chat_wrapper">
  <h2>AwesomeCo Help!</h2>
  <form id="nick_form" action="#" method="post" accept-
  charset="utf-8">
    <p>
      <label>Nickname
        <input id="nickname" type="text" value="GuestUser"/>
      </label>
      <input type="submit" value="Change">
    </p>
  </form>

  <div id="chat">connecting....</div>

  <form id="chat_form" action="#" method="post" accept-
  charset="utf-8">
    <p>
      <label>Message
        <input id="message" type="text" />
      </label>
      <input type="submit" value="Send">
    </p>
  </form>
</div>

</body>
</html>
```

В файл *chat.js* необходимо внести единственное изменение: задать переменную, определяющую местонахождение файла *WebSocketMain*.

html5_websockets/public/chat.js

```
WEB_SOCKET_SWF_LOCATION = "websocket_js/WebSocketMain.swf";
```

Когда все это будет сделано, наш чат-клиент будет работать во всех основных браузерах — при условии, что сервер, на котором размещается чат-сервер, также предоставляет файл с политикой сокетов Flash.

Политика сокетов Flash

По соображениям безопасности Flash Player взаимодействует через сокеты только с серверами, разрешающими подключения к Flash Player. Flash Player пытается загрузить файл политики сокетов сначала через порт 843, а затем через тот же порт, который используется вашим сервером. Предполагается, что сервер вернет ответ следующего вида:

```
<cross-domain-policy>
  <allow-access-from domain="*" to-ports="*" />
</cross-domain-policy>
```

Этот файл политики позволяет подключаться к сервису всем желающим. При работе с более конфиденциальными данными обычно устанавливаются ограничения доступа. Не забывайте, что файл должен предоставляться тем же сервером, на котором работает сервер Web Sockets, — через тот же порт или через порт 843.

В примерах кода этого раздела имеется простой сервер политики сокетов Flash, написанный на Ruby, который может использоваться для тестирования. За информацией о его настройке для тестирования в вашей среде обращайтесь к разделу «Серверы» (см. ниже).

Впрочем, чат-сервер — это только начало. Технология Web Sockets предоставляет мощный и простой механизм передачи данных в браузеры посетителей.

Серверы

В архив исходного кода книги включена версия сервера Web Sockets, для которой мы пишем клиента. Она написана на Ruby, поэтому вам понадобится интерпретатор Ruby. Информация о настройке Ruby для вашей системы приведена в файле *RUBY_README.txt* в исходном коде книги.

Чтобы запустить сервер, перейдите в папку, в которой он находится, и введите следующую команду:

```
ruby server.rb
```

Кроме чат-сервера в процессе тестирования примеров этой главы вам могут пригодиться два других сервера. Первый, *client.rb*, предоставляет интерфейс чата и файлы JavaScript. Второй, *flashpolicyserver*, предоставляет файл политики Flash, который используется кодом обходного решения для подключения к чат-серверу. Flash Player по файлам политики проверяет возможность взаимодействия с удаленным доменом.

Пользователи операционных систем Mac или Linux могут запустить все серверы командой

```
rake start
```

из папки *html5_websockets*.

Рецепт 26. Определение местоположения: Geolocation

Геопозиционированием называется метод определения текущего местоположения пользователя, а вернее его компьютера. Конечно, «компьютером» может быть не только настольная система, но и смартфон, планшетный компьютер или другое мобильное устройство. Текущее местоположение определяется на основании IP-адреса, MAC-адреса, координат точки доступа WiFi и даже координат GPS, если они доступны. Хотя технология Geolocation формально не является частью спецификации HTML5, она часто ассоциируется с HTML5, потому что две спецификации появились практически одновременно. В отличие от Web Storage технология Geolocation никогда не входила в спецификацию HTML. С другой стороны, как и Web Storage, эта чрезвычайно полезная технология была реализована в Firefox, Safari и Chrome. Рассмотрим пример ее использования.

Поиск центров поддержки

Нам поручено создать страницу контактов для сайта фирмы AwesomeCo. Директор по информационным технологиям спросил, сможем ли мы построить карту с текущим местонахождением посетителя и различных центров поддержки AwesomeCo. Ему не терпится увидеть прототип, поэтому работу нужно завершить как можно скорее.

Для решения задачи мы воспользуемся сервисом Google Static Map API, потому что он не требует ключа API, а мы собираемся построить очень простую карту.

Сервис-центры AwesomeCo расположены в Портленде, штат Орегон; Чикаго, штат Иллинойс; и в Провиденс, штат Род-Айленд. Static Map API позволяет относительно легко нанести эти точки на карту. Для этого достаточно построить тег `img` и включить адреса в URL.

html5geo/index.html

```
<img id="map" alt="Map of AwesomeCo Service Center locations"
src="http://maps.google.com/maps/api/staticmap?
&amp;size=900x300
&amp;sensor=false
&amp;maptype=roadmap
&amp;markers=color:green|label:A|1+Davol+square,+Providence,
     +RI+02906-3810
```

```
&amp;markers=color:green|label:B|22+Southwest+3rd+Avenue,
     Portland,+OR
&amp;markers=color:green|label:C|77+West+Wacker+Drive+Chicago
     +IL">
```

Сначала мы определяем размер изображения, а затем сообщаем Maps API, что передаваемая карте информация не используется сенсорным устройством (например, GPS-навигатором или клиентскими средствами геопозиционирования). Далее определяются маркеры на карте с указанием их меток и адресов. Данные маркеров также можно было бы передать в виде пар координат, разделенных запятыми, но в нашем демонстрационном приложении проще использовать этот вариант.

Определение местоположения посетителя

Также на карте необходимо обозначить текущее местоположение посетителя. Для этого мы определим на карте еще один маркер с текущей широтой и долготой. Необходимые данные запрашиваются у браузера следующим образом.

html5geo/index.html

```
navigator.geolocation.getCurrentPosition(function(position) {
  showLocation(position.coords.latitude,
  position.coords.longitude);
});
```

Метод предлагает пользователю предоставить свои координаты для отображения на карте. Если пользователь согласится, мы вызываем метод `showLocation()`.

Этот метод получает широту и долготу и строит карту заново, присоединяя новые данные к текущему источнику. Реализация метода выглядит так.

html5geo/index.html

```
1 var showLocation = function(lat, lng){
2  var fragment = "&markers=color:red|color:red|label:Y|" + lat
                  + "," + lng;
3  var image = $("#map");
4  var source = image.attr("src") + fragment;
5  source = source.replace("sensor=false", "sensor=true");
6  image.attr("src", source);
7 };
```

Вместо того чтобы дублировать исходный код всего изображения, мы присоединяем к нему широту и долготу новой точки. Перед тем как сохранять измененный источник в свойстве документа, мы меняем значение параметра `sensor` с `false` на `true` (метод `replace()` в строке 5).

При просмотре карты в браузере новая точка помечается буквой «Y». Пример карты показан на рис. 10.3.

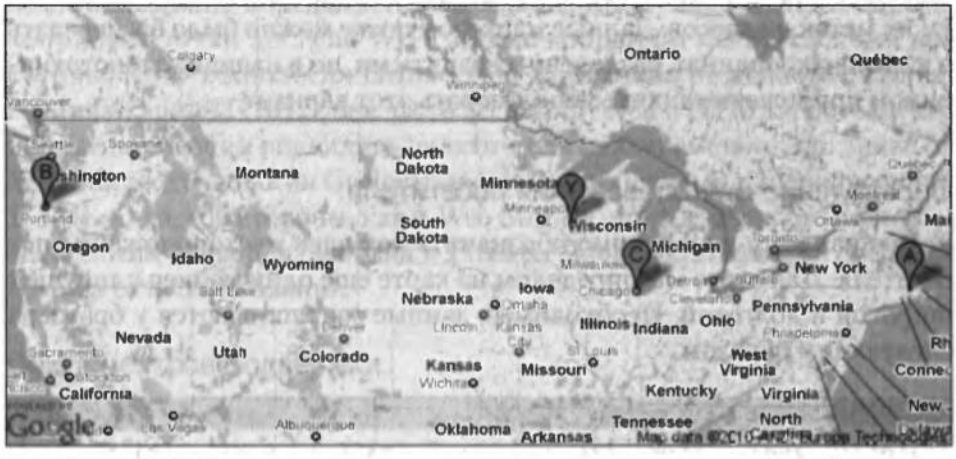


Рис. 10.3. Текущее местоположение посетителя помечается буквой «Y»

Обходное решение

На данном этапе посетители страницы будут видеть карту с обозначениями центров поддержки AwesomeCo, но при попытке загрузить страницу мы получим ошибку JavaScript. Прежде чем пытаться получать данные местоположения пользователя, необходимо проверить поддержку Geolocation.

html5geo/index.html

```
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function(position)
{
      showLocation(position.coords.latitude,
                   position.coords.longitude);
    });
  }else{
};
```

Google Ajax API[1] с возможностью определения местонахождения компьютера будет отличным обходным решением для нашей ситуации. Чтобы использовать его на сайте, необходимо получить ключ API, но для локального тестирования ключ не понадобится[2].

Наше обходное решение выглядит так.

html5geo/index.html

```
 1  var key = "your_key";
-  var script = "http://www.google.com/jsapi?key=" + key;
-  $.getScript(script, function(){
-    if ((typeof google == 'object') &&
 5      google.loader && google.loader.ClientLocation) {
-       showLocation(google.loader.ClientLocation.latitude,
-                    google.loader.ClientLocation.longitude);
-    }else{
-      var message = $("<p>Couldn't find your address.</p>");
10    message.insertAfter(«#map");
-    };
-  });
```

Метод jQuery `getScript()` используется для загрузки Google Ajax API. Далее метод `ClientLocation()` в строке 5 получает данные о местоположении посетителя, а вызов метода `showLocation()` наносит координаты на карту.

К сожалению, Google не может преобразовать некоторые IP-адреса в координаты, поэтому может оказаться, что вывести местоположение пользователя на карте невозможно; в этом случае под картой выводится соответствующее сообщение (строка 9). Наше обходное решение не идеально, но оно повышает вероятность успешного обнаружения посетителя.

Если вы не располагаете надежным методом получения координат от клиента, можно просто запросить у него адрес, но эту возможность при желании вы можете реализовать самостоятельно.

Перспективы

Технологии, описанные в этой главе, формально не являются частью HTML5, однако они представляют будущее веб-разработки. Со вре-

---

[1] http://code.google.com/apis/ajax/documentation/#ClientLocation

[2] Ключ также понадобится в случае размещения на http://localhost/. Для получения ключа можно обратиться по адресу http://code.google.com/apis/ajaxsearch/signup.html

менем все большая часть функциональности будет перемещаться на сторону клиента. Расширенное управление историей просмотра сделает клиентские приложения и Ajax-приложения более понятными и логичными. Технология Web Sockets заменяет периодический опрос удаленных служб отображением данных в реальном времени. Технология Cross-Document Messaging обеспечивает взаимодействие между веб-приложениями, невозможное в обычных условиях, а технология Geolocation со временем поможет строить веб-приложения, ориентированные на географическое расположение пользователя, а следовательно, все более актуальные с учетом расширения рынка мобильных компьютерных сред.

Исследуйте возможности этих технологий и следите за их распространением. Возможно, скоро они начнут играть важную роль в вашем инструментарии веб-разработки.

11 Что дальше?

Основное внимание в книге уделяется тем инструментам, которыми вы можете пользоваться прямо сейчас. Однако существуют и другие возможности, которые станут доступными в ближайшем будущем, — от поддержки 3D-графики на элементе `canvas` средствами WebGL до новых API хранения данных, переходов CSS3 и встроенной поддержки перетаскивания. В этой главе рассматриваются некоторые технологии недалекого будущего, которые уже могут использоваться как минимум в одном браузере, но еще не имеют достаточно хороших обходных решений (или попросту недостаточно четко определенные для того, чтобы работать с ними немедленно)[1]:

Переходы CSS3

Анимации при взаимодействиях с элементами. [C3, S3.2, F4, O10.5, IOS3.2, A2]

Web Workers

Фоновая обработка в JavaScript. [C3, S4, F3.5, O10.6]

3D-графика с использованием WebGL

Вывод 3D-объектов на холсте. [C5, F4]

IndexedDB

Механизм хранения пар «ключ/значение» на стороне клиента по аналогии с решениями NoSQL. [F4]

---

[1] В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

Перетаскивание

API перетаскивания. [C3, S4, F3.5, IE6, A2]

Проверка данных на формах

Проверка введенных данных на стороне клиента . [C5, S5, 10.6]

Начнем с переходов CSS3 и их использования в браузерах WebKit.

11.1. Переходы CSS3

Визуальные «подсказки» играют важную роль в хорошо спроектированном интерфейсе. CSS уже довольно давно поддерживает псевдокласс `:hover` для создания простейших «подсказок». Следующая разметка CSS оформляет ссылку так, чтобы она выглядела как кнопка.

css3transitions/style.css

```
a.button{
  padding: 10px;
  border: 1px solid #000;
  text-decoration: none;
}
a.button:hover{
  background-color: #bbb;
  color: #fff
}
```

Если навести указатель мыши на кнопку, цвет фона меняется с белого на серый, а цвет текста — с черного на белый. Переходы CSS3[1] позволяют сделать больше, включая создание простых анимаций, которые раньше могли быть реализованы только средствами JavaScript. Например, переход может реализовать эффект затухания; для этого в определение стиля включается следующий фрагмент:

css3transitions/style.css

```
1  a.button{
-    padding: 10px;
-    border: 1px solid #000;
-    text-decoration: none;
5    -webkit-transition-property: background-color, color;
-    -webkit-transition-duration: 1s;
```

[1] http://dev.w3.org/csswg/css3-transitions/

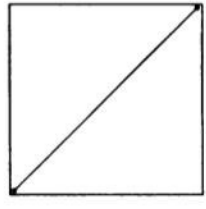
```
-     -webkit-transition-timing-function: ease-out;
-   }
-
10
-   a.button:hover{
-     background-color: #bbb;
-     color: #fff
-   }
```

В строке 5 мы указываем, к каким свойствам применяется переход. В данном случае изменяются два цвета, основной и фоновый. Продолжительность анимации задается в строке 6, а временна́я функция перехода — в строке 7.

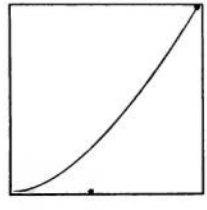
Временная функция

Свойство `transition-timing-function` описывает хронометраж перехода в рамках заданной продолжительности. Временная функция задается кубической кривой Безье, которая определяется четырьмя контрольными точками на графе. Каждая точка имеет координаты X и Y в интервале от 0 до 1. Первая и последняя контрольные точки всегда имеют координаты (0.0,0.0) и (1.0,1.0), а форма кривой определяется двумя средними точками.

Если средние контрольные точки совпадают с двумя граничными, кривая Безье принимает вид отрезка, наклоненного под углом в 45°. В этом случае четыре точки имеют координаты ((0.0, 0.0), (0.0,0.0), (1.0, 1.0), (1.0, 1.0)), а график выглядит так.

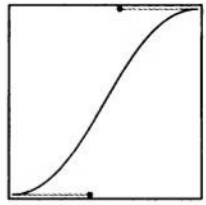


На следующем рисунке изображена более сложная кривая с точками ((0.0, 0.0), (0.42,0.0), (1.0, 1.0), (1.0, 1.0)).



Изменилось положение только второй контрольной точки, соответственно изгиб появляется только в левой нижней части кривой.

Еще более сложная кривая имеет изгибы и в верхней, и в нижней части.



Она определяется точками ((0.0, 0.0), (0.42,0.0), (0.58, 1.0), (1.0, 1.0)).

Форму кривой можно задать как значениями координат прямо в свойстве CSS, так и воспользоваться готовыми определениями, как было сделано в нашем примере. Вы можете использовать стандартные значения `default`, `ease-in`, `ease-out`, `ease-in-out`, `ease-out-in` и `cubic-bezier` (в этом случае вы сами задаете координаты точек).

Чтобы переход происходил с постоянной скоростью, используйте линейную зависимость. Со значением `ease-in` анимация начинается медленно, а потом ускоряется. Если вы хотите поближе познакомиться с кривыми Безье, в открытом доступе имеется отличный сценарий[1] с демонстрацией примеров и координат контрольных точек.

Поэкспериментируйте с переходами, но помните, что интерфейс должен быть прежде всего удобным, а уже затем красивым. Не создавайте переходы, которые раздражают пользователя мерцанием или слишком большой продолжительностью. Также стоит поближе познакомиться с анимацией CSS3[2] — другим механизмом изменения свойств CSS со временем.

11.2. Web Workers

Технология Web Workers[3] не входит в спецификацию HTML5, но, возможно, она пригодится вам для организации фоновых вычислений на стороне клиента, так что о ней стоит упомянуть особо.

---

[1] http://www.netzgesta.de/dev/cubic-bezier-timing-function.html

[2] http://www.w3.org/TR/css3-animations/

[3] http://www.whatwg.org/specs/web-workers/current-work/

Весь код на стороне клиента пишется на JavaScript, однако этот язык является однопоточным — он не позволяет выполнять сразу несколько операций одновременно. Если операция занимает много времени, пользователю придется дожидаться ее завершения. Для решения этой проблемы технология Web Workers реализует простой механизм параллельного выполнения.

Допустим, сценарий *worker.js*, занимающийся обработкой графики, можно запустить следующим образом:

webworkers/application.js

```
var worker = new Worker("worker.js");
```

Любой файл JavaScript можно запустить подобным образом, но чтобы выполнение шло независимо, запущенный сценарий не может обращаться к DOM. Таким образом, прямые манипуляции с элементами исключаются.

Основной сценарий может отправлять сообщения рабочему сценарию (worker script) при помощи функции `postMessage()`.

webworkers/application.js

```
$("#button").click(function(event){
  $("#output").html("starting...");
  worker.postMessage("start");
});
```

Рабочий сценарий также может отправлять сообщения основной странице, используя тот же метод `postmessage()`.

webworkers/worker.js

```
onmessage = function(event) {
  if(event.data === "start"){
    // Длинный цикл. Лучше заняться чем-нибудь более интересным.
    for (var x = 1; x <= 100000; x++){
      postMessage(x);
    }
  }
};
```

Для обработки этих сообщений мы прослушиваем событие `onmessage` в основном сценарии. Каждый раз, когда рабочий сценарий отправляет сообщение, инициируется выполнение следующего кода:

webworkers/application.js

```
worker.onmessage = function(event){
  $("#output").html(event.data);
}
```

По принципу работы этот API напоминает API междоменной передачи информации, который рассматривался на с. 213. В Internet Explorer технология Web Workers не поддерживается, поэтому вам придется положиться на Google Chrome Frame. Впрочем, если вам потребуется выполнить более серьезную работу на стороне клиента без блокировки, тему стоит исследовать глубже.

11.3. Встроенная поддержка перетаскивания

Возможность перетаскивания элементов интерфейса уже давно поддерживалась на уровне библиотек JavaScript, но недавно комитет W3C принял реализацию перетаскивания фирмы Microsoft как часть спецификации HTML5[1]. Она поддерживается Firefox, Safari, Internet Explorer и Chrome, но в действительности проблем хватает.

Реализация на первый взгляд кажется тривиальной: сначала мы помечаем элемент как «перетаскиваемый», после чего выбираем другой элемент для отслеживания «сброса» объекта; когда это происходит, выполняется некоторый фрагмент кода. К сожалению, на практике все не так просто. Для демонстрации давайте создадим простой интерфейс: при перетаскивании миниатюры в поле загружается полноразмерная версия фотографии.

html5drag/index.html

```
<div id="images">
  <img src="images/red_thumb.jpg"
    data-large="images/red.jpg" alt="A red flower">
  <img src="images/purple_thumb.jpg"
    data-large="images/purple.jpg"
    alt="A white and purple flower">
  <img src="images/white_thumb.jpg"
    data-large="images/white.jpg" alt="A white flower">

</div>
```

[1] http://dev.w3.org/html5/spec/dnd.html#dnd

```
<div id="preview">
  <p>Drop images here</p>
</div>
```

Для хранения ссылки на исходное изображение используется пользовательский атрибут данных.

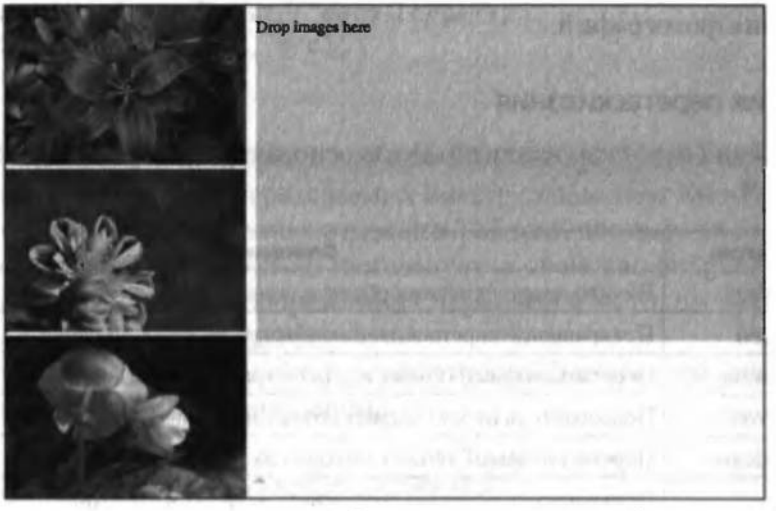


Рис. 11.1. Интерфейс просмотра фотографий

Определим базовые стили для создания двухстолбцового интерфейса.

html5drag/style.css

```
#images img{
  -webkit-user-drag
}

#images{
  float: left;
  width: 240px;
  margin-right: 10px;
}

#preview{
  float: left;
  width: 500px;
  background-color: #ddd;
  height: 335px;
}
```

```
.hover{
  border: 10px solid #000;
  background-color: #bbb !important;
}
```

С этим оформлением интерфейс выглядит так, как показано на рис. 11.1. Теперь необходимо добавить обработку событий для перетаскивания фотографий.

События перетаскивания

Реализация перетаскивания объектов основана на обработке нескольких событий.

Событие	Описание
ondragstart	Начало перетаскивания объекта пользователем
ondragend	Прекращение перетаскивания объекта *по любой причине*
ondragenter	Перетаскиваемый объект входит в границы приемника
ondragover	Пользователь перетаскивает объект над приемником
ondreagleave	Перетаскиваемый объект выходит за границы приемника
ondrop	Пользователь успешно сбрасывает элемент в приемник
ondrag	Пользователь перетаскивает объект в произвольное место; срабатывает многократно, но может использоваться для получения координат X и Y указателя мыши

Итак, с обработкой перетаскивания связано семь событий, и некоторые из событий имеют реализацию по умолчанию. Если не переопределить ее, вся схема перестает работать.

Прежде всего необходимо разрешить перетаскивание элементов списка:

html5drag/application.js

```
var contacts = $('#images img');
contacts.attr('draggable', 'true');
```

Мы добавляем атрибут HTML5 `draggable`. Это можно было сделать и в разметке, но поскольку реализация базируется на коде JavaScript, мы применяем этот атрибут в сценарии.

При перетаскивании изображения следует получить адрес полноразмерного изображения и сохранить его. Для этой цели будет использовано

событие ondragstart, а чтобы код был простым и кросс-платформенным, мы воспользуемся методом jQuery bind()[1].

html5drag/application.js

```
1 contacts.bind('dragstart', function(event) {
2   var data = event.originalEvent.dataTransfer;
3   var src = $(this).attr("data-large");
4   data.setData("Text", src);
5   return true;
6 });
```

В спецификации приведен механизм dataStorage для описания типа данных и самих данных, передаваемых вместе с событием. Метод jQuery bind() упаковывает событие в отдельный объект, поэтому для обращения к «настоящему» событию используется свойство originalevent (строка 2). URL-адрес изображения сохраняется методом setData() (строка 4), с типом данных Text.

Итак, теперь элементы можно перетаскивать. Рассмотрим инициирование событий при их сбрасывании.

Сбрасывание перетаскиваемых объектов

Приемником перетаскивания должно стать поле формы «То». Мы находим его и подключаем обработчик события drop.

html5drag/application.js

```
 1  var target = $('#preview');
-
-  target.bind('drop', function(event) {
-    var data = event.originalEvent.dataTransfer;
 5    var src = ( data.getData('Text') );
-
-    var img = $("<img></img>").attr("src", src);
-    $(this).html(img);
-    if (event.preventDefault) event.preventDefault();
10     return(false);
-  });
```

Адрес изображения, переданный в событии, читается методом getData() в строке 5. Далее мы создаем новый графический элемент, который используется для заполнения области контента.

---

[1] Не забывайте, что при использовании этого метода префиксы on опускаются.

Событие ondrop по умолчанию необходимо отменить, чтобы оно не инициировалось при сбрасывании перетаскиваемого элемента над приемником. Для этого необходимо вызвать `preventdefault()` и вернуть `false`. Internet Explorer требует возвращения `false`, а все остальные браузеры — вызова `preventDefault()`.

Однако при попытке выполнить этот код в Chrome или Safari он будет работать не совсем корректно. Как минимум необходимо переопределить событие ondragover; без этого наше событие ondrag выполняться не будет. Это делается следующим фрагментом кода.

html5drag/application.js

```
target.bind('dragover', function(event) {
  if (event.preventDefault) event.preventDefault();
  return false;
});
```

Здесь мы просто отменяем событие по умолчанию точно так же, как поступили с событием ondrop. То же самое необходимо сделать и с событием ondragend.

html5drag/application.js

```
contacts.bind('dragend', function(event) {
  if (event.preventDefault) event.preventDefault();
  return false;
});
```

Этот фрагмент отменяет все события браузера, инициируемые при прекращении перетаскивания элемента, но не мешает определенному нами событию ondrop.

Изменение стилей

Чтобы пользователь знал, когда перетаскиваемый элемент находится в границах приемника, мы воспользуемся методами ondragenter и ondragleave.

html5drag/application.js

```
target.bind('dragenter', function(event) {
  $(this).addClass('hover');
  if (event.preventDefault) event.preventDefault();
  return false;
});
```

```
target.bind('dragleave', function(event) {
  $(this).removeClass('hover');
  if (event.preventDefault) event.preventDefault();
  return false;
});
```

Класс `hover` из нашей таблицы стилей будет соответствующим образом применяться и отменяться при инициировании этих событий.

Перетаскивание файлов

Перемещение текста и элементов по странице — это только начало. Спецификация позволяет создавать интерфейсы с возможностью получения файлов с компьютера пользователя. Отправка фотографии или включение файла в сообщение сводится к простому перетаскиванию файла на заданный приемник. Кстати говоря, почтовый сервис Google Gmail поддерживает эту возможность в Firefox 3.6 и Chrome 5.

За дополнительной информацией по этой теме обращайтесь к превосходной статье[1] Лесли Майкла Орчарда.

Проблемы

Поведение нашего кода в разных браузерах выглядит, мягко говоря, непоследовательно. В IE8 решение работает, но при попытке изменить тип данных `setData()` с `Text` на `Url` работать перестает.

Кроме того, для поддержки перетаскивания элементов, не являющихся графическими изображениями или ссылками в Safari 4, в таблицу стилей придется добавить дополнительную директиву CSS.

```
#contents li{
  -webkit-user-drag
}
```

В этой книге неоднократно говорилось о том, как важно отделять стилевое оформление и поведение от контента; приведенный фрагмент прямо противоречит этой концепции.

Не пытайтесь перетаскивать текст на поля форм. Современные браузеры уже позволяют это делать, но нормального способа переопределения этого поведения не существует.

---

[1] http://decafbad.com/blog/2009/07/15/html5-drag-and-drop

При нынешнем положении намного лучших результатов с меньшим объемом кода можно добиться при помощи библиотеки JavaScript с поддержкой перетаскивания — такой, как jQuery UI[1].

Даже при использовании библиотеки остается еще одна проблема: доступность. В спецификации ничего не сказано о том, как быть с пользователями, которые не могут работать с мышью. Если в вашем интерфейсе реализована функциональность перетаскивания, необходимо разработать дополнительный метод, для работы которого не требуется JavaScript или мышь.

Спецификация обладает хорошим потенциалом, но в ней также остается немало нерешенных проблем. Используйте ее, если это уместно, но не заставляйте пользователей делать то, что делать неудобно или невозможно.

11.4. WebGL

В книге рассматривались возможности элемента `canvas` в контексте 2D-графики, но сейчас идет работа над другой спецификацией, ориентированной на работу с 3D-объектами. Спецификация WebGL[2] не является частью HTML5, однако в ее рабочую группу входят Apple, Google, Opera и Mozilla; они реализовали частичную поддержку спецификации в своих браузерах.

К сожалению, работа с 3D-графикой выходит за рамки книги. Превосходные примеры и учебники по этой теме можно найти на сайте Learning WebGL[3].

11.5. Indexed Database API

В книге рассматривались два механизма хранения данных на стороне клиента: Web Storage и Web SQL Storage. Фонд Mozilla возражает против спецификации Web SQL; по его мнению, спецификация не должна базироваться на конкретном ядре SQL. Так появилась новая спецификация

---

[1] http://docs.jquery.com/UI/Draggable

[2] https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/doc/spec/WebGL-spec.html

[3] http://learningwebgl.com/blog/?p=11

Indexed Database API, которая, как запланировано, станет отдельным стандартом[1].

Механизм Indexed Database API предназначен для хранения пар «ключ/значение» (по аналогии с такими инструментами Web Storage, как `localStorage` и `sessionStorage`), но он предусматривает возможность построения нетривиальных запросов. К сожалению, на момент написания книги доступных реализаций этой спецификации не было, так что описывать какие-либо подробности бессмысленно, скорее всего, они изменятся к моменту появления реализаций. Предполагается, что спецификация будет поддерживаться в Firefox 4 и Chrome 7.

За развитием этой спецификации стоит внимательно следить, потому что технология Web SQL находится в тупике, а фонд Mozilla неоднократно заявлял, что не собирается реализовывать Web SQL в Firefox, потому что ему не нравится диалект SQL, а спецификация не должна базироваться на одной конкретной реализации базы данных. Технология Web SQL использует диалект SQLite, который может изменяться независимо от спецификации. Весьма вероятно, что спецификация также будет реализована и в Internet Explorer, так как фирма Microsoft проявляла интерес к ее разработке[2].

11.6. Проверка данных форм на стороне клиента

В спецификации HTML5 представлены атрибуты, которые могут использоваться для проверки данных, введенных пользователем, на стороне клиента, чтобы простые ошибки ввода обнаруживались до отправки запросов на сервер. Эта задача традиционно решалась на уровне JavaScript, но формы HTML5 могут использовать новые атрибуты для определения поведения.

Например, чтобы объявить поле формы обязательным для заполнения, добавьте атрибут `required`:

`html5validation/index.html`

```
<label for="name">Name</label>
<input type="text" name="name" autofocus required id="name">
```

---

[1] http://www.w3.org/TR/IndexedDB/

[2] http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/

Если поле останется незаполненным, то вместо отправки браузер выведет сообщение об ошибке; такая проверка не требует ни единой строки кода JavaScript. Она уже поддерживается в Opera, как показано на рис. 11.2.

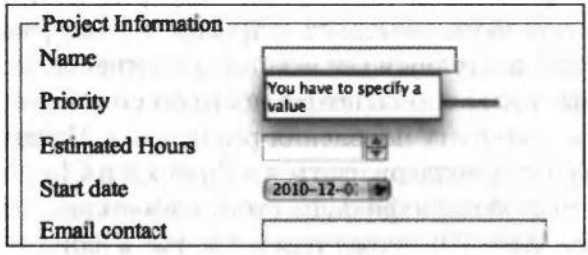


Рис. 11.2. Предупреждение в Opera

Таким образом, пользователи сразу узнают об ошибках ввода, не дожидаясь ответа сервера. Проверка может быть отключена, недоступна или попросту неправильно реализована, поэтому вы должны иметь в резерве стратегию проверки данных на стороне сервера. Однако и в этом случае атрибуты помогут найти поля, обязательные для заполнения, и оформить интерфейс средствами CSS так, чтобы эти поля визуально отличались от остальных.

Атрибут `pattern` позволяет пойти еще дальше и определить регулярное выражение для проверки соответствия содержимого поля заданному критерию.

html5validation/index.html

```
<input type="text"
       name="acctnumber" id="acctnumber"
       required
       pattern="^[1-9]+[0-9]*$" >
```

В настоящее время ни один браузер не поддерживает эту возможность в своем интерфейсе, однако готовую разметку удобно использовать для проверки данных средствами библиотек JavaScript.

11.7. Вперед!

Разработчиков ждут интересные времена. Эта книга дает лишь общее представление о ближайших тенденциях веб-разработки. В специфи-

кациях содержится намного больше полезной информации и с ними определенно стоит познакомиться поближе. Надеюсь, после того что вы здесь узнали, вы займетесь самостоятельной работой и изучением различных спецификаций в ходе этой работы.

А теперь — за дело! Начинайте создавать новые потрясающие веб-приложения!

IV

Приложения

Приложение A

Краткий справочник

В дальнейших описаниях поддержка тех или иных возможностей браузерами обозначается сокращенным названием браузера и минимальным номером версии в квадратных скобках. Используются следующие сокращения C: Google Chrome, F: Firefox, IE: Internet Explorer, O: Opera, S: Safari, IOS: устройства iOS с Mobile Safari, A: Android Browser.

A.1. Новые элементы

См. рецепт 1, «Реструктуризация блога с использованием семантической разметки», с. 28.

`<header>`

Определение заголовка страницы или раздела. [C5, F3.6, IE8, S4, O10]

`<footer>`

Определение завершителя страницы или раздела. [C5, F3.6, IE8, S4, O10]

`<nav>`

Определение области навигации страницы или раздела. [C5, F3.6, IE8, S4, O10]

`<section>`

Определение логической области страницы или группировка контента. [C5, F3.6, IE8, S4, O10]

`<article>`

Определение статьи (логически завершенного блока контента). [C5, F3.6, IE8, S4, O10]

`<aside>`

Определение вторичного или связанного контента. [C5, F3.6, IE8, S4, O10]

`<meter>`

Представление величины, находящейся в заданном диапазоне. [C5, F3.5, S4, O10]

`<progress>`

Отображение информации о ходе некоторой операции в реальном времени. [Не поддерживался на момент издания книги.]

A.2. Атрибуты

Пользовательские атрибуты данных

Возможность включения пользовательских данных в любой элемент с использованием схемы `data-`. [Все браузеры поддерживают чтение таких данных методом JavaScript `getAttribute()`]

См. рецепт 2, «Создание всплывающих окон с пользовательскими атрибутами данных», с. 42.

Редактирование «на месте» `<p contenteditable>lorem ipsum</p>`

Поддержка редактирования контента «на месте» в браузере. [C4, S3.2, IE6, O10.1]

См. рецепт 6, «Редактирование „на месте“», с. 67.

A.3. Формы

См. рецепт 3, «Описание данных при помощи новых полей», с. 50.

`<input type="email">`

Поле для ввода адреса электронной почты. [O10.1, IOS]

`<input type="url">`

Поле для ввода URL-адреса. [O10.1, IOS]

`<input type="tel">`

Поле для ввода телефонного номера. [O10.1, IOS]

`<input type="search">`

Поле для ввода ключевых слов поиска. [C5, S4, O10.1, IOS]

```
<input type="range">
```

Ползунок. [C5, S4, O10.1]

```
<input type="number">
```

Поле для ввода числовых данных, часто в виде элемента-счетчика. [C5, S5, O10.1, IOS]

```
<input type="date">
```

Поле для ввода даты. Поддерживаются типы `date`, `month` и `week`. [C5, S5, O10.1]

```
<input type="datetime">
```

Поле для ввода даты со временем. Поддерживаются типы `datetime`, `datetime-local` и `time`. [C5, S5, O10.1]

```
<input type="color">
```

Поле для выбора цвета. [C5, S5] (Chrome 5 и Safari 5 «понимают» поле `Color`, но не отображают конкретный элемент).

A.4. Атрибуты полей форм

```
<input type="text" autofocus>
```

Поддержка передачи фокуса конкретному элементу формы. [C5, S4]

См. рецепт 4, «Использование автофокуса для перехода к первому полю», с. 58.

```
<input type="email" placeholder="me@example.com">
```

Поддержка автоматического включения текста в поле формы. [C5, S4, F4]

См. рецепт 5, «Заполняющий текст», с. 60.

`required` [`<input type="email" required >`]

Объявление поля формы, обязательного для заполнения. [C5, S5, O10.6]

См. раздел 11.6, «Проверка данных форм на стороне клиента», с. 245.

`pattern` [`<input type="text" pattern="^[1-9]+[0-9]*$">`]

Проверка данных поля формы на соответствие заданному регулярному выражению. [C5, S5, O10.6]

См. раздел 11.6, «Проверка данных форм на стороне клиента», с. 245.

А.5. Доступность

Атрибут `role` [`<div role="document">`]

Описание обязанностей элементов для экранных дикторов. [C3, F3.6, S4, IE8, O9.6]

См. рецепт 11, «Роли ARIA и упрощение навигации», с. 102.

`aria-live` [`<div aria-live="polite">`]

Идентификация автоматически обновляемых (вероятно, средствами Ajax) областей. [F3.6 (Windows), S4, IE8]

См. рецепт 12, «Создание обновляемых областей с улучшенной доступностью», с. 107.

`aria-atomic` [`<div aria-live="polite" aria-atomic="true">`]

Признак чтения всего контента активной области или только изменившихся элементов. [F3.6 (Windows), S4, IE8]

См. рецепт 12, «Создание обновляемых областей с улучшенной доступностью», с. 107.

А.6. Мультимедиа

`<canvas>` [`<canvas id="my_canvas" width="150" height="150">`]

Создание векторной графики из кода JavaScript. [C4, F3, IE9, S3.2, O10.1, IOS3.2, A2]

См. рецепт 13, «Рисование логотипа», с. 117.

`<audio>` [`<audio src="drums.mp3"></audio>`]

Воспроизведение аудио встроенными средствами браузера. [C4, F3.6, IE9, S3.2, O10.1, IOS3, A2]

См. рецепт 15, «Работа с аудио», с. 140.

`<video>` [`<video src="tutorial.m4v"></video>`]

Воспроизведение видео встроенными средствами браузера. [C4, F3.6, IE9, S3.2, O10.5, IOS3, A2]

См. рецепт 16, «Внедрение видео», с. 144.

A.7. CSS3

См. раздел 11.1, «Переходы CSS3», с. 234.

`:nth-of-type [p:nth-of-type(2n+1){color: red;}]`

Поиск всех *n* элементов определенного типа. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 7, «Стилевое оформление таблиц с использованием псевдоклассов», с. 77.

`:first-child [p:first-child{color:blue;}]`

Поиск первого дочернего элемента. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 7, «Стилевое оформление таблиц с использованием псевдоклассов», с. 77.

`:nth-child [p:nth-child(2n+1){color: red;}]`

Поиск заданного дочернего элемента в прямом направлении. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 7, «Стилевое оформление таблиц с использованием псевдоклассов», с. 77.

`:last-child [p:last-child{color:blue;}]`

Поиск последнего дочернего элемента. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 7, «Стилевое оформление таблиц с использованием псевдоклассов», с. 77.

`:nth-last-child [p:nth-last-child(2){color: red;}]`

Поиск заданного дочернего элемента в обратном направлении. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 7, «Стилевое оформление таблиц с использованием псевдоклассов», с. 77.

`:first-of-type [p:first-of-type{color:blue;}]`

Поиск первого элемента заданного типа. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 7, «Стилевое оформление таблиц с использованием псевдоклассов», с. 77.

`:last-of-type [p:last-of-type{color:blue;}]`

Поиск последнего элемента заданного типа. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

Поддержка столбцов:

`[#content{ column-count: 2; column-gap: 20px; column-rule: 1px solid #ddccb5; }]`

Разбиение области контента на несколько столбцов. [C2, F3.5, S3, O9.5, IOS3, A2]

См. рецепт 9, «Создание многостолбцовых макетов», с. 91.

`:after [span.weight:after { content: "lbs"; color: #bbb; }]`

Используется с `content` для вставки контента после заданного элемента. [C2, F3.5, S3, IE8, O9.5, IOS3, A2]

См. рецепт 8, «Печать ссылок (:after)», с. 87.

Медиа-запросы `[media="only all and (max-width: 480)"]`

Применение стилей в зависимости от параметров устройства. [C3, F3.5, S4, IE9, O10.1, IOS3, A2]

См. рецепт 10, «Построение мобильных интерфейсов», с. 97.

`border-radius [border-radius: 10px;]`

Закругление углов элементов. [C4, F3, IE9, S3.2, O10.5]

См. рецепт 17, «Закругление прямых углов», с. 154.

Поддержка RGBa `[background-color: rgba(255,0,0,0.5);]`

Использование цветов RGB вместо шестнадцатеричных кодов (с альфа-каналом). [C4, F3.5, IE9, S3.2, O10.1]

См. рецепт 18, «Тени, градиенты и преобразования», с. 162.

`box-shadow [box-shadow: 10px 10px 5px #333;]`

Создание теней, отбрасываемых элементами. [C3, F3.5, IE9, S3.2, O10.5]

См. рецепт 18, «Тени, градиенты и преобразования», с. 162.

Повороты `[transform: rotate(7.5deg);]`

Поворот любого элемента. [C3, F3.5, IE9, S3.2, O10.5]

См. рецепт 18, «Тени, градиенты и преобразования», с. 162.

Градиенты `[linear-gradient(top, #fff, #efefef);]`

Построение градиента для использования в качестве графического изображения. [C4, F3.5, S4]

См. рецепт 18, «Тени, градиенты и преобразования», с. 162.

`@font-face [@font-face { font-family: AwesomeFont; ] src: url(http://example.com/awesomeco.ttf); font-weight: bold; }]`

Возможность использования конкретных шрифтов в CSS. [C4, F3.5, IE5+, S3.2, O10.1]

См. рецепт 19, «Использование шрифтов», с. 173.

A.8. Хранение данных на стороне клиента

`localStorage`

Хранение данных в виде пар «ключ/значение» с привязкой к домену и сохранением данных между сеансами. [C5, F3.5, S4, IE8, O10.5, IOS, A]

См. рецепт 20, «Сохранение настроек с использованием localStorage», с. 184.

`sessionStorage`

Хранение данных в виде пар «ключ/значение» с привязкой к домену и уничтожением данных при завершении сеанса. [C5, F3.5, S4, IE8, O10.5, IOS, A]

См. рецепт 20, «Сохранение настроек с использованием localStorage», с. 184.

Web SQL Databases

Полноценные реляционные базы данных с поддержкой создания таблиц, вставки, обновления, удаления, выборки и транзакций, с привязкой к домену и сохранением данных между сеансами. [C5, S3.2, O10.5, IOS3.2, A2]

См. рецепт 21, «Хранение данных в реляционной базе данных на стороне клиента», с. 191.

A.9. Другие API

Offline Web Applications

Кэширование файлов для автономного использования; позволяет приложениям работать без подключения к Интернету. [C4, S4, F3.5, O10.6, IOS3.2, A2]

См. рецепт 22, «Автономная работа», с. 204.

History

Управление историей просмотра. [C5, S4, IE8, F3, O10.1 IOS3.2, A2]

См. рецепт 23, «История просмотра», с. 209.

Cross-Document Messaging

Передача сообщений между окнами с контентом, загруженным в разных доменах. [C5, S5, F4, IOS4.1, A2]

См. рецепт 24, «Передача информации между доменами», с. 213.

Web Sockets

Создание подключения с поддержкой состояния между браузером и сервером. [C5, S5, F4, IOS4.2]

См. рецепт 25, «Чат на базе Web Sockets», с. 220

Geolocation

Получение информации о широте и долготе. [C5, S5, F3.5, O10.6, IOS3.2, A2]

См. рецепт 26, «Определение местоположения: Geolocation», с. 228.

Web Workers

Фоновая обработка в JavaScript. [C3, S4, F3.5, O10.6]

См. раздел 11.2, «11.2 Web Workers», с. 236.

3D-графика с использованием WebGL

Вывод 3D-объектов на холсте. [C5, F4]

См. раздел 11.4, «11.4 WebGL», с. 244.

Перетаскивание

API перетаскивания. [C3, S4, F3.5, IE6, A2]

См. раздел 11.3, «Встроенная поддержка перетаскивания», с. 238.

Приложение Б

Введение в jQuery

Написание кода JavaScript, который бы четко и однозначно работал во всех основных браузерах, — весьма непростая задача. Существует много библиотек, которые упрощают этот процесс; одной из самых популярных среди них является jQuery. Она проста в использовании, обладает обширной базой готового кода и хорошо подходит для создания простых обходных решений.

В этом приложении представлены основные компоненты библиотеки jQuery, которые используются во многих местах книги. Настоящее приложение не заменит превосходно написанной документации jQuery[1] и не содержит полного списка всех возможностей и методов. Тем не менее оно станет хорошей отправной точкой для самостоятельного изучения.

Б.1. Загрузка jQuery

Вы можете загрузить библиотеку с сайта jQuery[2] и связать ее со сценарием jQuery напрямую, но в книге мы загружаем jQuery с серверов Google:

```
<script type="text/javascript"
  charset="utf-8"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.
min.js">
</script>
```

Количество одновременных подключений к серверу из браузера ограничено. Если распределить графику и сценарии по нескольким серверам,

---

[1] http://docs.jquery.com

[2] http://www.jquery.com

это ускорит загрузку страниц пользователями. Использование сети доставки контента Google также имеет дополнительное преимущество — так как другие сайты связываются с библиотекой jQuery в Google, она может уже находиться в кэше их браузеров (как вам, вероятно, известно, для идентификации кэшированной копии браузер использует полный URL-адрес файла). С другой стороны, если вы собираетесь работать с jQuery на портативном или настольном компьютере, не имеющем постоянного подключения к Интернету, используйте локальную копию.

Б.2. Основы jQuery

После того как библиотека jQuery будет загружена вашей страницей, можно начинать работать с элементами. jQuery содержит функцию с именем `jQuery()`; эта функция является «ядром» библиотеки. В книге она используется для выборки элементов по селекторам CSS и их упаковки в объекты jQuery для последующей обработки. Также существует сокращенная версия функции `jQuery()` с именем `$()`;, именно она используется в книге. В оставшейся части настоящего приложения я буду использовать термин «функция jQuery». Рассмотрим пару примеров ее использования.

Для поиска тега `h1` на странице используется запись вида

```
$("h1");
```

Если потребуется найти все элементы с классом `important`, используйте запись

```
$(".important");
```

Еще раз посмотрите на эти два примера. Они различаются только селектором CSS. Функция jQuery возвращает объект jQuery — специальный объект JavaScript с массивом элементов DOM, соответствующих селектору. Объект определяет много полезных методов для выполнения операций с отобранными элементами. Рассмотрим некоторые из них более подробно.

Б.3. Методы изменения контента

В этом разделе перечислены методы, используемые для изменения контента HTML в примерах книги.

Методы hide и show

Методы `hide()` и `show()` скрывают и отображают элементы пользовательского интерфейса. Чтобы скрыть один или несколько элементов на странице, используйте запись следующего вида:

```
$("h1").hide();
```

Для отображения скрытых элементов достаточно вызвать метод `show()`. Метод `hide()` часто используется в книге для сокрытия разделов страниц, которые должны отображаться только при отключении JavaScript (например, расшифровок аудироликов или другого обходного контента).

Методы html, val и attr

Метод `html()` используется для чтения и записи внутреннего контента заданного элемента.

```
$("message").html("Hello World!");
```

В этом примере между открывающим и закрывающим тегами `h1` записывается текст «Hello World».

Метод `val()` предназначен для чтения и записи значений полей форм. Он работает практически так же, как метод `html()`.

Метод `attr()` предназначен для чтения и записи атрибутов элементов.

Методы append, prepend и wrap

Метод `append()` добавляет новый дочерний элемент после существующих элементов. Допустим, имеется простая форма и пустой неупорядоченный список.

```
<form id="add">
  <label for="task">Task</label>
  <input type="text" id="task" >
  <input type="submit" value="Add">
</form>
<ul id="links">
</ul>
```

Новые элементы списка можно создавать, присоединяя их при отправке формы.

```
$(function(){
  $("#add").submit(function(event){
    event.preventDefault();
    var new_element = $("<li>" + $("#email").val() + "</li>");
    $("#links").append(new_element);
  });
});
```

Метод `prepend()` работает по аналогии с методом `append()`, но новые элементы вставляются не после существующих, а перед ними. Метод `wrap()` «упаковывает» выбранный элемент в элемент, представленный заданным объектом jQuery.

```
var wrapper = $("#message").wrap("<div><h2>Message</h2></div>").
parent();
```

Используя эти методы, можно на программном уровне создавать достаточно сложные структуры.

CSS и классы

Метод `css()` используется для определения стилей элементов.

```
$(«label").css(«color", «#f00");
```

Также можно воспользоваться синтаксисом хеша JavaScript для применения к элементу сразу нескольких правил CSS:

```
$("h1").css( {"color" : "red",
              "text-decoration" : "underline"}
           );
```

Впрочем, подобное смешение стилевого оформления со сценарным кодом нежелательно. Методы jQuery `addClass()` и `removeClass()` могут использоваться для добавления и удаления классов при возникновении определенных событий; стили лучше ассоциировать с этими классами. Например, события jQuery в сочетании с классами позволяют изменять фон полей формы при получении и потере ими фокуса:

```
$("input").focus(function(event){
  $(this).addClass("focused");
});

$("input").blur(function(event){
  $(this).removeClass("focused");
});
```

Этот тривиальный пример можно заменить псевдоклассом CSS3 `:focus`, но в некоторых браузерах этот псевдокласс не поддерживается.

Сцепленные вызовы

Методы объектов jQuery возвращают объекты jQuery; следовательно, вызовы методов можно сцеплять до произвольной длины.

```
$("h2").addClass("hidden").removeClass("visible");
```

Однако злоупотреблять сцепленными вызовами не рекомендуется, потому что они затрудняют чтение кода.

Б.4. Создание элементов

Время от времени требуется создать новый элемент HTML для его вставки в документ. Для создания элементов можно воспользоваться методом `jQuery()`.

```
var input = $("input");
```

Хотя того же результата можно добиться вызовом `document.createElement("input");`, использование функции jQuery упрощает дополнительные вызовы методов.

```
var element = $("<p>Hello World</p>");
element.css("color", "#f00").insertAfter("#header");
```

Этот пример в очередной раз показывает, как сцепленные вызовы jQuery помогают создавать структуры и выполнять различные операции с ними.

Б.5. События

Часто при взаимодействии пользователя со страницей должны инициироваться разные события. В jQuery многие распространенные события представляют собой обычные методы объекта jQuery с параметром-функцией. Например, чтобы все ссылки на странице с классом `popup` открывались в новом окне, выполните следующий фрагмент:

```
1  var links = $("#links a");
2  links.click(function(event){
3    var address = $(this).attr('href');
```

```
4     event.preventDefault();
5     window.open(address);
6   });
```

В обработчике события jQuery для обращения к текущему элементу используется ключевое слово `this`. В строке 3 оно передается функции jQuery, где для него вызывается метод `attr()` с целью быстрого получения адреса ссылки.

Функция `preventDefault()` предотвращает инициирование исходного события, чтобы оно не мешало нашей обработке.

Привязка

Некоторые события не поддерживаются jQuery напрямую; для их обработки можно воспользоваться методом `bind()`. Например, при реализации перетаскивания из спецификации HTML5 необходимо отменить событие `ondragover`. Метод `bind()` используется следующим образом.

```
target = $(«#droparea")
target.bind('dragover', function(event) {
  if (event.preventDefault) event.preventDefault();
  return false;
});
```

Обратите внимание: для обрабатываемого события префикс `on` не указывается.

Исходное событие

При использовании событийных функций jQuery (таких, как `bind()` или `click()`) jQuery инкапсулирует событие JavaScript в отдельном объекте, копируя в него лишь часть исходных свойств. Иногда требуется получить доступ к исходному событию для обращения к свойствам, которые не были скопированы. События jQuery предоставляют доступ к исходному событию через свойство `originalEvent`. Обращение к свойству `data` события `onmessage` выглядит следующим образом:

```
$(window).bind("message",function(event){
  var message_data = event.originalEvent.data;
});
```

Свойство `originalEvent` может использоваться для обращения к любым свойствам и методам исходного события.

Б.6. Функция document.ready

Выражением «ненавязчивый JavaScript» обозначается код JavaScript, полностью отделенный от контента. Вместо включения атрибутов `onclick` в элементы HTML мы используем обработчики событий вроде тех, которые рассматривались в разделе Б.5. Поведение добавляется в документ без изменения самого документа, а разметка HTML не зависит от того, включена ли пользовательская поддержка JavaScript.

Недостаток такого подхода заключается в том, что JavaScript «не видит» никакие элементы документа до их объявления. Код JavaScript можно было бы включить в блок `script` в конце страницы, но такое решение препятствует повторному использованию кода между страницами. Код можно было бы включить в обработчик события JavaScript `window.onLoad()`, но событие инициируется после загрузки всего контента. Задержка означает, что пользователи будут взаимодействовать с элементами до подключения к ним событий. Нам понадобится механизм добавления событий при загрузке DOM, но до отображения страницы.

Функция jQuery `document.ready` решает именно эту задачу, притом это решение работает во всех браузерах. Пример ее использования:

```
$(document).ready(function() {
  alert("Hi! I am a popup that displays when the page loads");
});
```

Также существует более компактная версия, которую мы используем в своем коде.

```
$(function() {
  alert("Hi! I am a popup that displays when the page loads");
});
```

Эта схема используется почти в каждом примере книги для простого и ненавязчивого включения обходных решений в проекты.

Мы рассмотрели лишь малую часть того, что можно сделать при помощи jQuery. Кроме манипуляций с документом, jQuery предоставляет методы сериализации форм, создания запросов Ajax, а также ряд вспомогательных функций, упрощающих перебор и перемещение по модели DOM. Несомненно, по мере освоения jQuery вы найдете много других возможностей для использования этой библиотеки в своих проектах.

Приложение Г

Ресурсы

Apple—HTML5: *http://www.apple.com/html5/*

Страница фирмы Apple о поддержке HTML5 и веб-стандартов в браузере Safari 5.

CSS3.Info: *http://www.css3.info/*

Большое количество справочных материалов и примеров по различным модулям спецификации CSS3.

Font Squirrel: *http://www.fontsquirrel.com*

Бесплатные шрифты в различных форматах, подходящие для распространения в Web.

HTML5: *http://www.w3.org/TR/html5/*

Спецификация HTML5 на сайте W3C.

HTML5 – Mozilla Developer Center: *https://developer.mozilla.org/en/html/html5*

Страница HTML5 на сайте Mozilla Developer Center.

Реализация серверов Web Sockets с использованием Node.js: *http://www.web2media.net/laktek/2010/05/04/implementing-web-socket-servers-with-node-js/*

Написание серверов Web Sockets с использованием Node.js.

Microsoft IE9 Test-Drive: *http://ie.microsoft.com/testdrive/*

Демонстрация возможностей HTML5 (и другой сопутствующей функциональности) в Internet Explorer 9.

Ruby и Web Sockets: *http://www.igvita.com/2009/12/22/ruby-websockets-tcp-for-the-browser/*

Информация о em-websocket – библиотеке Ruby для построения серверов Web Sockets.

Файлы политики сокетов Flash: *http://www.lightsphere.com/dev/articles/flash_socket_policy.html*

Подробное описание файлов политики сокетов Flash.

Typekit: *http://www.typekit.com*

Сервис, позволяющий использовать лицензированные шрифты на сайтах с использованием простого JavaScript API.

Unit Interactive: «Стеки шрифтов CSS»: *http://unitinteractive.com/blog/2008/06/26/better-css-font-stacks/*

Обсуждение стеков шрифтов с превосходными примерами.

Видео для всех! *http://camendesign.com/code/video_for_everybody*

Информация о поддержке видео в HTML5 с примерами кода воспроизведения видео во всех браузерах.

Video.js: *http://videojs.com*

Библиотека JavaScript, упрощающая воспроизведение видео HTML5.

Когда можно использовать... *http://caniuse.com/*

Таблицы совместимости браузеров для HTML5, CSS3 и других сопутствующих технологий.

Приложение

Библиография

[Hog09] Brian P. Hogan. *Web Design For Developers*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2009.

[HT00] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Reading, MA, 2000.

[Zel09] Jeffrey Zeldman. *Designing With Web Standards*. New Riders Press, New York, third edition, 2009.

полезная информация, но вам придется поэкспериментировать с выбором настроек, которые лучше всего подойдут для вас и ваших слушателей.

B.2 Кодирование видео для Web

Чтобы видео HTML5 можно было просматривать на всех платформах, видеофайлы необходимо закодировать в нескольких форматах. Кодирование в форматах H.264, Theora и VP8 занимает слишком много времени — как при настройке кодировщиков с открытым кодом вроде FFMpeg[1], так и при непосредственном выполнении кодирования. Принципы кодирования видеоматериалов выходят за рамки книги. Мы не располагаем достаточным количеством страниц для объяснения следующей команды, преобразующей файл в формат VP8 с использованием контейнера WebM:

```
ffmpeg -i blur.mov
-f webm -vcodec libvpx_vp8 -acodec libvorbis
-ab 160000 -sameq
blur.webm
```

Если вы не хотите возиться с настройками самостоятельно, веб-сервис Zencoder[2] может преобразовать ваши видеофайлы во все форматы, необходимые для HTML5. Вы размещаете видеофайлы на Amazon S3 или на другом общедоступном URL-адресе, а затем создаете задания по кодированию видео в различные форматы через веб-интерфейс или вызовы API. Zencoder загружает видеофайлы, выполняет кодирование, а затем отправляет полученные материалы на ваши серверы. Сервис Zencoder не бесплатен, но он дает отличные результаты и экономит массу времени при большом объеме кодируемого контента[3].

Если вы хотите поэкспериментировать с форматами самостоятельно, к вашим услугам еще один хороший вариант: Miro Video Converter[4]. Программа имеет готовые наборы настроек для преобразования видеофайлов в разные форматы и распространяется с открытым кодом.

---

[1] http://www.ffmpeg.org/

[2] http://www.zencoder.com/

[3] Для полной ясности уточню, что я знаком с парой программистов из Zencoder, но я бы рекомендовал этот сервис в любом случае.

[4] http://mirovideoconverter.com/

Приложение B

Кодирование аудио и видео

Кодирование аудио- и видеоматериалов для использования с тегами HTML5 `audio` и `video` — сложная тема, выходящая за рамки книги. Тем не менее это короткое приложение хотя бы показывает правильное направление, если вам когда-либо придется заниматься подготовкой собственного контента.

B.1. Кодирование аудио

Чтобы ваше приложение было ориентировано на максимально широкую аудиторию, звуковые файлы необходимо подготовить в форматах MP3 и Vorbis, а для этого вам понадобится пара инструментов.

При кодировании файлов в формате MP3 лучшее качество обеспечивает кодек Lame. При кодировании следует использовать переменную скорость передачи информации (битрейт). Команда для выполнения качественного кодирования выглядит примерно так.

```
lame in.wav out.mp3 -V2 --vbr-new -q0 --lowpass 19.7
```

Для кодирования аудио в формате Vorbis используется программа Oggenc. Качественный файл Vorbis с переменной скоростью передачи кодируется командой вида:

```
oggenc -q 3 inputfile.wav
```

За дополнительной информацией о кодировании MP3 и Vorbis обращайтесь на сайт Hydrogen Audio[1]. Там представлена исключительно

[1] Lame — по адресу http://wiki.hydrogenaudio.org/index.php?title=Lame#Quick_start_.28short_answer.29; Vorbis — по адресу http://wiki.hydrogenaudio.org/index.php?title=Recommended_Ogg_Vorbis.o