Брайан Хоган

HTML5 и CSS3

Веб-разработка по стандартам нового поколения

2-е издание

HTML5 and CSS3

Level Up with Today's Web Technologies

Brian P. Hogan

2nd edition

The Pragmatic Bookshelf
Raleigh, North Carolina Dallas, Texas

БИБЛИОТЕКА ПРОГРАММИСТА

Брайан Хоган

HTML5 и CSS3

Веб-разработка по стандартам нового поколения

2-е издание

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2014

Брайан Хоган

HTML5 и CSS3. Веб-разработка по стандартам нового поколения. 2-е изд.

Серия «Библиотека программиста»

Перевел с английского Е. Матвеев

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

ББК 32.988.02-018
УДК 004.76838.5

Хоган Б.

Х68 HTML5 и CSS3. Веб-разработка по стандартам нового поколения. 2-е изд. — СПб.: Питер, 2014. — 320 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-00979-9

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.

Второе издание содержит обновленную информацию по использованию HTML5 и CSS3; здесь представлена более подробная информация о средствах доступности HTML5 и описан ряд новых подходов, обходных решений и рецептов.

12+ (Для детей старше 12 лет. В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-1937785598 англ.
ISBN 978-5-496-00979-9

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), д. 3, литер А, пом. 7Н.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 10.01.14. Формат 70x100/16. Усл. п. л. 25,800. Тираж 1700. Заказ № 23.
Отпечатано в полном соответствии с качеством предоставленных издательством материалов
в ГППО «Псковская областная типография». 180004, Псков, ул. Ротная, 34.

Краткое содержание

Содержание

Благодарности

Работа над вторым изданием обычно занимает немного времени — автор исправляет ошибки или вносит усовершенствования и обновления в первое издание. Но на этот раз все выглядело так, словно я написал новую книгу, и я хочу поблагодарить многих людей, существенно упростивших мою работу.

Прежде всего я хочу поблагодарить вас за то, что вы читаете эту книгу. Надеюсь, она поможет вам взяться за собственные интересные и впечатляющие проекты.

Замечательный коллектив издательства «The Pragmatic Bookshelf» заслуживает не только моей благодарности — он внес значительный вклад в создание книги. Сюзанна Пфальцер (Susannah Pfalzer) снова проследила за тем, чтобы мои тексты выглядели осмысленно. Она замечательный редактор, и я благодарен ей за потраченное время и внимание к деталям – особенно в такой книге (с тысячами мелких деталей, требующих внимания). Дэйв Томас (Dave Thomas) и Энди Хант (Andy Hunt) делились своим мнением, и я благодарен им за неустанную поддержку. Спасибо вам!

Мне повезло еще и в том, что над книгой работала совершенно потрясающая группа технических рецензентов. Комментарии и обратная связь были превосходными, исчерпывающими и полными полезных советов по улучшению материала. Спасибо всем, кто мне помогал: Шайен Кларк (Cheyenne Clark), Джоэл Клермон (Joel Clermont), Джон Кули (Jon Cooley), Чед Думлер-Монплезир (Chad Dumler-Montplaisir), Джефф Холланд (Jeff Holland), Майкл Хантер (Michael Hunter), Кэролайн Клевер (Karoline Klever), Стивен Орр (Stephen Orr), Дэн Риди (Dan Reedy), Лорен Сэндс-Рэмшоу (Loren Sands-Ramshaw), Брайан Шо (Brian Schau), Мэтью Джон Сайас (Matthew John Sias), Тибор Симик (Tibor Simic), Чарли Стрэн

(Charley Stran) и Колин Йейтс (Colin Yates). Ваши рецензии не только были подробными, но содержали замечательные советы и наблюдения, существенно повлиявшие на окончательную версию этой книги.

Спасибо Джессике Янюк (Jessica Janiuk) за предоставленные снимки экранов для устройств Android. Я также хочу поблагодарить своих деловых партнеров — Криса Уоррена (Chris Warren), Криса Джонсона (Chris Johnson), Майка Уэбера (Mike Weber), Ника ЛаМуро (Nick LaMuro), Остена Отта (Austen Ott), Эрика Тески (Erich Tesky), Кевина Гизи (Kevin Gisi) и Джона Кинни (Jon Kinney) — за их постоянную поддержку.

Наконец, моя жена Карисса изо всех сил следила за тем, чтобы я работал изо всех сил. Она — мой безмолвный соавтор, и я вечно благодарен ей за любовь и поддержку. Спасибо тебе, Карисса, за все, что ты для меня делаешь.

Предисловие

Для веб-разработчика три месяца в веб-программировании — все равно что год реального времени. А это означает, что с выхода предыдущего издания этой книги прошло 12 веб-лет.

Мы, веб-разработчики, постоянно слышим о чем-то новом. Еще несколько лет назад HTML5 и CSS3 казались делом будущего, но уже сегодня компании используют эти технологии в своей работе, потому что браузеры (Chrome, Safari, Firefox, Opera и Internet Explorer) начали реализовывать отдельные части этой спецификации.

Технологии HTML5 и CSS3 закладывают основу для мощных, интерактивных 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 дополнительные атрибуты, 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. Динамические клиентские приложения JavaScript станут более доступными для людей с ограниченными возможностями, 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 и клиентов. В некоторых случаях фрагменты спецификации CSS3 (например, тени, градиентные заливки и трансформации) называются «HTML». Создатели браузеров стараются превзойти друг друга в степени поддержки «HTML5» их продуктами. Заказчики начинают выдвигать странные требования: «Мой сайт должен быть написан на HTML5».

В основной части этой книги мы сосредоточимся на спецификациях HTML5 и CSS3 и на использовании описанных в них методов в основных браузерах. В завершающей части книги рассматривается группа сопутствующих спецификаций, которые прямо сейчас используются на разных платформах. К их числу относятся спецификации 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 книге есть несколько отдельных глав, посвященных исключительно CSS3, но материал, посвященный CSS3, встречается и в других главах.

Во многих разделах присутствует подраздел «Обходное решение», в котором рассказано, что делать, если браузер пользователя еще не поддерживает необходимые функции. Работоспособность обходных решений обеспечивается разными средствами, от сторонних библиотек до наших собственных решений на базе JavaScript и jQuery.

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

Книга открывается кратким обзором HTML5 и CSS3. Мы рассмотрим несколько новых структурных тегов, используемых для 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 в HTML5 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, переходами и анимациями.

Далее будут представлены клиентские средства HTML5 — Web Storage, IndexedDB 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.

Основное внимание в книге уделяется тому, что можно использовать уже сегодня в современных браузерах. Некоторые возможности HTML5 и CSS3 еще не получили широкого распространения, но выглядят достаточно перспективно. Они более подробно описаны в главе 11.

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.

Списки совместимости

В начале каждой главы приводится список возможностей HTML, рассматриваемых в этой главе. В этих списках поддержка браузеров 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 Safari, A: Android Browser.

Что в книге не рассматривается

Мы не будем рассматривать версии Internet Explorer, предшествующие Internet Explorer 8. Компания Microsoft активно отваживает пользователей от этих старых браузеров.

Также мы не будем рассматривать все без исключения аспекты HTML5 и CSS3. О некоторых вещах говорить просто бессмысленно, потому что их реализации изменились или еще не имеют практического применения. Например, сетчатые макеты CSS выглядят заманчиво[1], но на них не стоит тратить время, пока все браузеры не выйдут на один уровень. В этой книге я постарался показать, как использовать средства HTML5 и CSS прямо сейчас и с расчетом на самую широкую аудиторию.

Так как в книге отсутствует базовый материал по HTML и CSS, она написана, прежде всего, для веб-разработчиков с хорошим знанием HTML и CSS. Новичкам я бы порекомендовал сначала прочитать книгу Джона Дакетта *HTML and CSS: Design and Build Websites* [Duc11], в ней хорошо излагаются основы. Также стоит обратить внимание на книгу Джеффри Зельдмана *Designing with Web Standards* [Zel09].

Также предполагается, что читатель хотя бы в общих чертах разбирается в JavaScript и jQuery[2] (последняя технология будет использована для реализации многих обходных решений). Все основные методы jQuery, используемые в книге, рассматриваются в приложении Б, но читатель также может обратиться к другим источникам информации — например, к книге Кристофа Портнева *Pragmatic Guide to JavaScript* [Por10], содержащей более подробную справочную информацию о JavaScript. В последней части книги JavaScript используется довольно интенсивно, но я уверен, что вы справитесь.

Изменения во втором издании

Второе издание книги содержит обновленный материал; из него исключено все, что относилось непосредственно к Internet Explorer 7 и ниже. В ней представлена более подробная информация о средствах доступности HTML5, более надежные и проверенные обходные решения, а также девять новых рецептов:

- Рецепт 2, «Вывод информации о ходе выполнения операции с использованием элемента <meter>».

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

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

- Рецепт 4, «Применение списков описаний для определения FAQ».
- Рецепт 8, «Проверка пользовательского ввода без использования JavaScript».
- Рецепт 16, «Улучшение доступности таблиц».
- Рецепт 19, «Создание векторной графики SVG».
- Рецепт 22, «Доступность видео».
- Рецепт 26, «Переходы и анимации».
- Рецепт 28, «Хранение информации в базе данных на стороне клиента с использованием IndexedDB».
- Рецепт 34, «Перетаскивание».

Также в главе 11 мы рассмотрим модель CSS Flexible Box, междоменный доступ к ресурсам, фоновые вычисления (web workers), отправка событий серверами и эффекты фильтров CSS.

Помимо нового материала, в другие рецепты при необходимости были включены новые обходные решения. В архиве примеров кода вы найдете удобный веб-сервер на базе Node.js, который упростит тестирование всех проектов в разных браузерах.

Как читать книгу

Не думайте, будто вы обязаны прочитать книгу «от корки до корки». Материал разбит на доступные разделы, сосредоточенные на одной или двух базовых концепциях. В каждой главе представлено несколько проектов. Загрузив архив примеров кода с сайта книги[1], вы найдете в нем папку template/, которая станет хорошей отправной точкой для экспериментов.

В книге встречаются примеры кода следующего вида:

```
html5_new_tags/index.html
<link rel="stylesheet" href="stylesheets/style.css">
```

Метка над кодом показывает, где можно найти файл в коде примеров.

Наконец, разбирайте код, приведенный в книге, и не бойтесь экспериментировать с готовыми примерами. А теперь давайте выясним более конкретно, что же понадобится для работы с примерами.

[1] http://pragprog.com/titles/bhh52e/

Что вам понадобится

Для тестирования кода, приведенного в книге, потребуется браузер Firefox 20 и выше, Chrome 20 и выше, Opera 10.6 или Safari 6. А еще лучше иметь все эти браузеры, потому что каждый из них несколько отличается в нюансах от других. Устройство на базе Android или iOS тоже пригодится, но не является обязательным.

Тестирование в Internet Explorer

Также вам понадобится возможность тестирования сайтов в Internet Explorer 8 и выше для проверки работоспособности обходных решений. Проще всего эта задача решается установкой Microsoft Windows в VirtualBox[1] для тестирования. Компания Microsoft предоставляет бесплатные виртуальные машины для тестирования веб-приложений на сайте Modern.IE, где можно загрузить готовые образы для VirtualBox, Parallels или VMWare[2]. Эти машины работают тридцать дней, после чего их придется загружать заново.

Node.js и сервер примеров

Для тестирования некоторых возможностей в книге необходимо, чтобы файлы HTML и CSS поставлялись с веб-сервера, а тестирование других требует более сложной служебной части. В архиве примеров кода вы найдете сервер, который упрощает работу с примерами. Для запуска этого сервера необходимо установить Node.js по инструкциям, размещенным на сайте Node.js[3]. Для предотвращения периодических сбоев сервера следует установить версию не ниже 0.10.0.

Также вам понадобится программа командной строки *npm* (Node Packaged Modules) для установки зависимостей. Эта программа устанавливается в составе установки Node.js.

После установки Node.js посетите сайт книги и загрузите примеры кода. Распакуйте архив, перейдите к местонахождению распакованных файлов в терминале (или окне командной строки, если вы работаете в Windows) и выполните следующую команду (естественно, *без* `$`) для загрузки всех зависимостей:

```
$ npm install
```

[1] http://virtualbox.org

[2] http://modern.ie

[3] http://nodejs.org

Затем введите следующую команду (также без `$`):

```
$ node server
```

Команда запускает сервер на порте 8000. Загрузите в браузере страницу *http://localhost:8000* и просмотрите примеры. Если тестирование осуществляется на виртуальных машинах, ваши машины должны быть способны установить связь с фактическим IP-адресом компьютера, на котором работает сервер примеров. А самое замечательное, что все папки и файлы, находящиеся в одной папке с файлом сервера, будут автоматически предоставляться им, так что вы сможете работать с материалом книги, перемещаясь по папкам примеров.

Пара слов об использовании JavaScript и jQuery

В этой книге широко используется JavaScript. В прошлом файлы JavaScript часто загружались в секции `<head>` страницы, после чего различными средствами (такими, как метод jQuery `document.ready()`) приложение ожидало, пока модель DOM (Document Object Model) станет доступной для изменения. Однако сейчас рекомендуется загружать все сценарии в нижней части страницы, так как это улучшает быстродействие. Мы так и будем поступать. Все сценарии, включая jQuery, будут загружаться в конце страницы — за исключением немногочисленных ситуаций, в которых необходимо изменять DOM до загрузки всех элементов.

Кроме того, мы будем использовать jQuery там, где это разумно. Если нам потребуется просто найти элемент по идентификатору, мы воспользуемся `document.getElementById()`, но для обработки событий или более сложных манипуляций с DOM, которые должны работать в Internet Explorer 8, будет применяться jQuery.

Другими словами, мы будем «выбирать инструмент под конкретную задачу». Возможно, в отдельных случаях это будет выглядеть немного непоследовательно, но это неизбежно, когда мы начинаем вводить обходные решения для сохранения работоспособности приложения в старых браузерах. А я непременно объясню причины для выбора того или иного решения.

Сетевые ресурсы

На сайте книги[1] размещены ссылки на форумы, списки опечаток и ссылка на исходный код всех примеров.

[1] http://www.pragprog.com/titles/bhh52e/

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[1]. В нем будут публиковаться сопутствующие материалы, обновления и рабочие примеры.

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

От издательства

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

Мы будем рады узнать ваше мнение!

[1] http://www.beyondhtml5andcss3.com/

Обзор HTML5 и CSS3

HTML5 и CSS3 — 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 этих стандартов, 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.

Обратная совместимость

Одна из самых веских причин для немедленного перехода на HTML5 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 целевыми браузерами, но документ будет проверяться на валидность как разметка HTML5.

Более содержательная разметка

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

Меньше хлама

Многие элементы HTML5 были упрощены, и для них были определены более разумные значения по умолчанию. Вы уже видели, насколько проще стал элемент `doctype`, но это далеко не единственный пример. Скажем, нас годами учили, что тег JavaScript `<script>` должен определяться следующим образом:

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

Но в HTML5 предполагается, что вы будете использовать JavaScript во всех тегах `<script>`, так что эти дополнительные атрибуты смело можно опустить.

Если вы хотите указать, что документ содержит символы UTF-8, достаточно воспользоваться тегом

```
<meta charset="utf-8">
```

вместо громоздкого и часто копируемого тега

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

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.

ВОПРОС/ОТВЕТ

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

Конечно, сможете! Посмотрите на разметку Polyglot Markup[1]. 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

[1] http://www.w3.org/TR/html-polyglot/

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. Следите за тем, чтобы они выдавались с правильным типом MIME. За дополнительной информацией об этой и других проблемах обращайтесь по адресу http://www.webdevout.net/articles/beware-of-xhtml#myths.

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. Завершив-

тели, боковые панели и другие структурные элементы легко перемещаются или вовсе исключаются из просмотра. Упрощение разбора страниц в целом также облегчит 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 вы научитесь использовать эти новые атрибуты, чтобы современные экранные дикторы могли работать с ними.

Расширенные селекторы

Селекторы CSS3 позволяют легко идентифицировать 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.

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 31, «Передача информации между доменами».

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 32, «Чат на базе Web Sockets».

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, «Хранение данных на стороне клиента».

1.2. Тернистый путь в будущее

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

Старые версии Internet Explorer

Internet Explorer в настоящее время имеет самую большую пользовательскую базу, а версии 8 и ниже имеют очень слабую поддержку HTML5 и CSS3. IE10 существенно улучшает ситуацию, но эта версия еще не

получила широкого распространения, и она останется недоступной для пользователей Windows Vista и более ранних операционных систем. 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. В книге вы увидите много тактических приемов такого рода.

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 изображений в документах HTML средствами JavaScript. Это создает проблемы не только для лиц с ограниченными зрительными возможностями, но и для 5 % пользователей Интернета, отключающих JavaScript в своих браузерах[1].

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 — для тех, кто ее любит.

Стремясь 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.

[1] http://visualrevenue.com/blog/2007/08/eu-and-us-javascript-disabled-index.html

Устаревшие теги

В HTML5 появилось много новых элементов, но спецификация также объявляет устаревшими некоторые стандартные элементы, которые могут присутствовать в ваших веб-страницах[1]. 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`.

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

[1] http://www.w3.org/TR/html5-diff/

приложения, вы сможете добиться того же эффекта при помощи инструментов, предоставленных вашими средствами веб-разработки. Например, обратите внимание на свойство CSS `position:fixed`.

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`.

Атрибут `profile` тега `head` в дальнейшем поддерживаться не будет. Учтите, что он часто встречается в шаблонах WordPress.

Наконец, исчез атрибут `longdesc` элементов `<img>` и `<iframe>`. Вероятно, это огорчит разработчиков, которым приходится решать проблемы доступности, потому что в атрибуте `longdesc` было принято передавать дополнительные описания для экранных дикторов.

Если вы собираетесь использовать HTML5 на существующем сайте, найдите эти элементы и либо удалите их, либо замените семантическими аналогами. Обязательно проверьте валидность своих страниц службой W3C Validator[1] — она поможет в поиске устаревших тегов и атрибутов.

Борьба корпоративных интересов

Internet Explorer — не единственный браузер, создатели которого не торопятся с переходом на HTML5 и CSS3. У Google, Apple и Mozilla тоже есть

[1] http://validator.w3.org/

свои интересы, и они ведут борьбу за лидерство. В настоящее время они спорят относительно поддержки видео- и аудиокодеков, а свои мнения выражают в очередных версиях браузеров. Например, элемент `<audio>` в Safari воспроизводит аудио в формате MP3, но файлы в формате OGG в нем не работают. Напротив, Firefox поддерживает файлы OGG вместо MP3.

Со временем эти противоречия будут разрешены. А пока приходится принимать разумные решения относительно поддержки — либо ограничивая реализацию браузерами, используемыми целевой аудиторией, либо многократно реализуя одну и ту же функциональность для разных браузеров, пока не выйдет финальная версия стандартов. Все это не так страшно, как кажется на первый взгляд. Более подробная информация приводится в главе 7.

Работа над HTML5 и CSS3 продолжается

Спецификации еще не утверждены окончательно, а это означает, что все сказанное в них может измениться. Хотя в Firefox, Chrome и Safari реализована сильная поддержка HTML5, в случае изменения спецификации браузеры тоже изменятся; это может привести к появлению устаревших, неработоспособных сайтов. Например, в последние годы 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 HTML5 и CSS3 и оставаться в курсе происходящего, все будет хорошо. Спецификация HTML5 доступна по адресу *http://www.w3.org/TR/html5/*. Спецификация CSS3 разбита на несколько модулей, и вы можете проследить за ее прогрессом по адресу *http://www.w3.org/Style/CSS/current-work*.

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

В начальных главах этой книги мы поговорим о том, как использовать возможности HTML5 и CSS3 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`.

2 Новые структурные теги и атрибуты

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="page">
  <div id="navbar_wrapper">
    <div id="navbar">
      <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/products">Products</a></li>
        ...
      </ul>
    </div>
  </div>
</div>
```

Здесь неупорядоченный список, уже являющийся блочным элементом, упакован в два тега `div`, которые также являются блочными элементами. Напомним, что блочные элементы всегда начинаются с новой строки, тогда как встроенные (inline) элементы не требуют разрыва строки, так что тег

 оказывается лишним. 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.

Кроме новых структурных тегов мы также рассмотрим такие теги, как <meter> и <progress>, и возможности использования пользовательских атрибутов HTML5, позволяющих встраивать данные прямо в элементы. Короче говоря, речь пойдет о том, как выбрать правильный тег для конкретной задачи. Взяв на вооружение HTML5, мы сможем избавиться от симптомов дивита на всю оставшуюся жизнь.

В этой главе рассматриваются следующие новые элементы и возможности:

<header>
: Определение заголовка страницы или раздела. [C5, F3.6, S4, IE8, O10]

<footer>
: Определение завершителя страницы или раздела. [C5, F3.6, S4, IE8, O10]

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

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

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

<aside>
: Определение вторичного или связанного контента. [C5, F3.6, S4, IE8, O10]

Списки описаний

Определение списка имен и связанных с ними значений (таких, как определения или описания). [Все браузеры]

`<meter>`

Представление величины, находящейся в заданном диапазоне. [C8, F16, S6, O11]

`<progress>`

Отображение информации о ходе некоторой операции в реальном времени. [C8, F6, S6, IE10, O11]

Пользовательские атрибуты данных

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

Рецепт 1. Реструктуризация блога с использованием семантической разметки

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.

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. 1. Готовый макет

На рис. 2 изображена общая схема страницы. Она имеет вполне типичную для блогов структуру: основная заголовочная область с главным заголовком и расположенной под ним горизонтальной областью навигации. Вглавном 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.

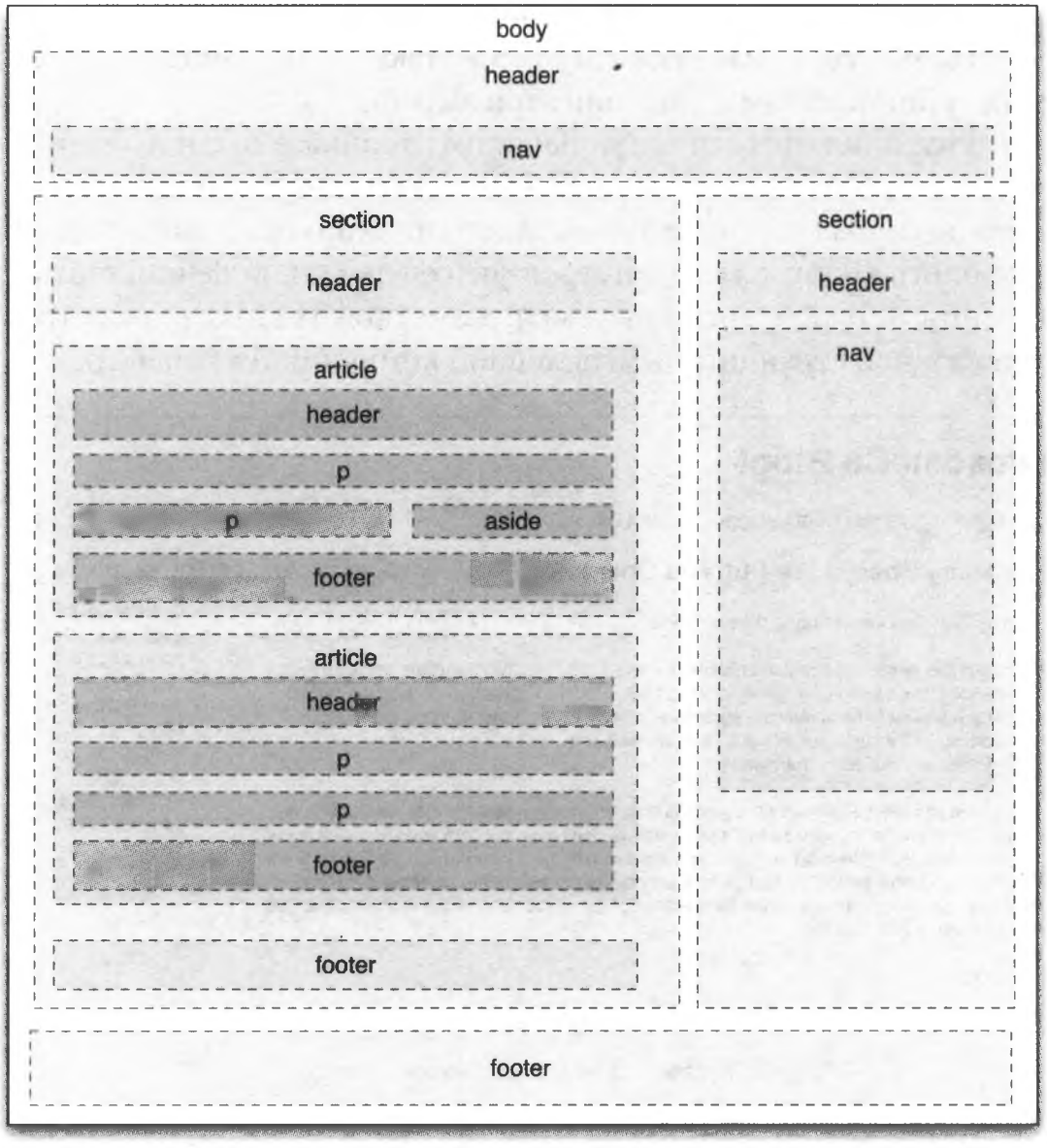


Рис. 2. Структура блога в семантической разметке HTML5

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:

html5_new_tags/index.html

```
1  <!DOCTYPE html>
2  <html lang="en-US">
3    <head>
4      <meta 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.

Обратите внимание на тег `<meta>` в строке 4. Он определяет кодировку символов страницы. Если мы хотим использовать символы Юникода, этот тег необходимо включить в самое начало страницы — до первой строки, содержащей текст.

Базовый шаблон готов, теперь можно переходить к созданию блога.

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.

html5_new_tags/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.

Завершители

Тег <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 (хотя также встречаются завершители со сложной навигационной структурой). 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.

html5_new_tags/index.html

```
<footer id="page_footer">
  <p>Copyright © 2013 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.

Область навигации

Навигация играет исключительно важную роль в успехе сайта. Люди попросту не задержатся на вашем сайте, если им будет слишком сложно найти искомую информацию, поэтому область навигации заслуживает собственного тега HTML.

Включим область навигации в заголовок нашего документа. В ней будут размещены ссылки на домашнюю страницу блога, архивы, страницу со списком соавторов блога и страницу с контактными данными.

Страница может иметь несколько элементов навигации. Области навигации часто размещаются как в заголовке, так и в завершителе страницы; теперь вы можете явно сослаться на нужную область. В завершителе нашего блога размещаются ссылки на домашнюю страницу AwesomeCo, страницу с информацией о компании, а также ссылки на страницы с условиями обслуживания и политикой конфиденциальности. Мы объединим их в другой неупорядоченный список в элементе `footer` страницы.

html5_new_tags/index.html

```
<footer id="page_footer">
  <p>Copyright © 2013 AwesomeCo.</p>
  <nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Terms of Service</a></li>
      <li><a href="#">Privacy</a></li>
    </ul>
  </nav>
</footer>
```

Внешний вид обеих навигационных областей будет определяться средствами CSS, так что пока о нем можно не беспокоиться. Новые элементы предназначены для описания контента, а не для описания его внешнего вида; для этого существует CSS. Давайте продолжим работу над разметкой.

Разделы и статьи

Раздел (section) представляет собой логическую область страницы. Для описания логических разделов вместо тега `<div>`, которым злоупотребляли многие разработчики, следует использовать элемент `<section>`.

html5_new_tags/index.html

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

Впрочем, не увлекайтесь определением разделов. Используйте их для логической группировки контента! В приведенном примере создается раздел для хранения всех сообщений в блоге. Не стоит определять для каждого сообщения собственный раздел — для этого существует более подходящий тег.

Статьи

Тег `<article>` идеально подходит для описания основного информационного наполнения веб-страницы. При таком количестве элементов на странице (заголовки, завершители, навигационные элементы, реклама, виджеты, закладки социальных сетевых сервисов) легко забыть, что люди приходят на ваш сайт ради предоставляемого контента. Тег `<article>` поможет описать этот контент.

Чем же статьи отличаются от разделов? Раздел определяет логическую часть документа, а статья — его фактическое содержание (заметка, сообщение в блоге, новость и т. д.).

Раздел можно сравнить со спортивной рубрикой в газете: рубрика может состоять из нескольких статей, каждая из которых может существовать самостоятельно. При этом 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.

Каждая статья в нашем блоге будет состоять из заголовка, контента и завершителя. Статьи определяются следующим образом:

html5_new_tags/index.html

```
<article class="post">
  <header>
    <h2>How Many Should We Put You Down For?</h2>
    <p>Posted by Brian on
      <time datetime="2013-10-01T14:39">October 1st,
                                2013 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 &ldquo;Are you
    interested in our 3 or 5 year service plan?&rdquo; Instead,
    ask &ldquo;Are you interested in the 3 year service plan or
    the 5 year plan, which is a better value?&rdquo;
    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 &ldquo;no.&rdquo;
  </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>`.

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`.

html5_new_tags/index.html

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

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

Завершенный раздел вместе с дополнением выглядит так:

html5_new_tags/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="2013-10-01T14:39">October 1st, 2013 at
        2:39PM</time>
      </p>
    </header>
    <aside>
      <p>
        &ldquo;Never give someone a chance to say no when
        selling your product.&rdquo;
      </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 &ldquo;Are
      you interested in our 3 or 5 year service plan?&rdquo;
      Instead, ask &ldquo;Are you interested in the 3 year
      service plan or the 5 year plan, which is a better
      value?&rdquo;
      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 &ldquo;no.&rdquo;
    </p>
    <footer>
      <p><a href="comments"><i>25 Comments</i></a> ...</p>
    </footer>
  </article>
</section>
```

Осталось добавить раздел боковой панели.

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>.

html5_new_tags/index.html

```
<section id="sidebar">

  <nav>
    <h3>Archives</h3>

    <ul>
      <li><a href="2013/10">October 2013</a></li>
      <li><a href="2013/09">September 2013</a></li>
      <li><a href="2013/08">August 2013</a></li>
      <li><a href="2013/07">July 2013</a></li>
      <li><a href="2013/06">June 2013</a></li>
      <li><a href="2013/05">May 2013</a></li>
      <li><a href="2013/04">April 2013</a></li>
      <li><a href="2013/03">March 2013</a></li>
      <li><a href="2013/02">February 2013</a></li>
      <li><a href="2013/01">January 2013</a></li>
    </ul>

  </nav>

</section>
```

В нашем случае ссылки на боковой панели страницы формируют вторичный механизм навигации. Не каждая группа ссылок должна заключаться в элемент <nav>; этот элемент резервируется исключительно для областей навигации.

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 . Сначала мы создаем новый файл таблицы стилей *stylesheets/style.css* и присоединяем его к документу HTML, включая ссылку в заголовок.

html5_new_tags/index.html

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

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

html5_new_tags/stylesheets/style.css

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

p{ margin: 0 0 20px 0;}

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

Затем определим ширину заголовка.

html5_new_tags/stylesheets/style.css

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

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

html5_new_tags/stylesheets/style.css

```
#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{
  margin: 0 20px 0 0;
  padding:0;
  display:inline;
}
```

Мы добавляем небольшие поля справа от каждого элемента `<li>`, чтобы создать интервалы между командами меню. В сокращенной записи значения `margin` читаются в порядке «сверху-справа-снизу-слева» (как на циферблате, начиная с 12 часов и по часовой стрелке).

Затем к главному контенту будет применено стилевое оформление для создания большого столбца контента и меньшей боковой панели. Раздел `posts` размещается слева, а также задается его ширина; выноска в статье размещается справа. Раз уж мы занимаемся оформлением выноски, заодно увеличим размер шрифта.

html5_new_tags/style.css

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

#posts aside{
  float: right;
  font-size: 20px;
  line-height: 40px;
  margin-left: 5%;
  width: 35%;
}
```

Также необходимо разместить боковую панель и определить ее ширину.

html5_new_tags/stylesheets/style.css

```
#sidebar{
  float: left;
  width: 25%;
}
```

Наконец, осталось определить завершитель. Мы зададим для него атрибут `clear`, чтобы завершитель располагался в нижней части страницы. Вспомните: элемент с атрибутом `float` исключается из нормального потока макетирования документа. Назначение атрибута `clear` сообщает браузеру, что к элементу не следует применять «всплытие»[1].

[1] https://developer .mozilla.org/en-US/docs/Web/CSS/clear

html5_new_tags/stylesheets/style.css

```
#page_footer{
  clear: both;
  display: block;
  text-align: center;
  width: 100%;
}
```

В нашем примере используются только простейшие стили. Не сомневаюсь, что вы сможете придать ему намного, намного лучший вид.

Обходное решение

Наш пример отлично работает в Internet Explorer 9, Firefox, Chrome, Opera и Safari, но начальство вряд ли обрадуется при виде страницы в Internet Explorer 8. Контент отображается нормально, но поскольку Internet Explorer 8 не поддерживает использованные элементы, он не может применить к ним стили, и вся страница выглядит так, словно ее делали в середине 1990-х годов.

Как же заставить Internet Explorer 8 и более ранних версий применять стилевое оформление к этим элементам? Есть только один способ — использовать JavaScript для определения элементов как части документа. Сделать это совсем несложно. Мы включим соответствующий код в раздел `<head>` страницы, чтобы он был выполнен до отображения каких-либо элементов браузером. Код будет размещен в *условном комментарии* — особой разновидности комментариев, которые будут читаться только в Internet Explorer.

html5_new_tags/index.html

```
<!--[if lte IE 8]>
<script>
  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/

Рецепт 2. Вывод информации о ходе выполнения операции с использованием элемента <meter>

AwesomeCo проводит благотворительную кампанию по привлечению средств, в ходе которой рассчитывает собрать пожертвования в сумме $5000. Если нужная сумма будет собрана, руководство AwesomeCo великодушно пообещало добавить еще $5000. AwesomeCo хочет вывести на одной из своих страниц шкалу прогресса, которая должна выглядеть примерно так, как показано на рис. 3.

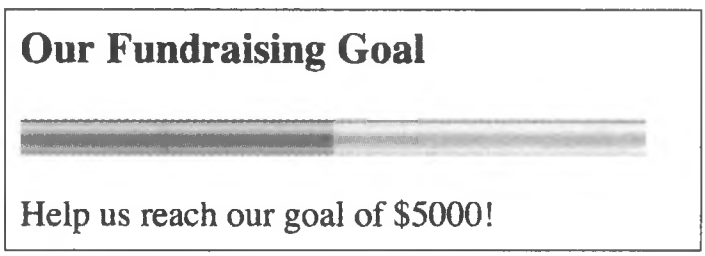


Рис. 3. Шкала прогресса с информацией о ходе сбора средств

Конечно, нужного результата можно добиться при помощи тегов `<div>`, к которым применяются стили CSS, но также можно воспользоваться новым тегом `<meter>`, разработанным специально для таких задач. Тег `<meter>` помогает создать семантическое описание шкалы. Чтобы ваша шкала соответствовала рекомендациям спецификации, не используйте ее для отображения данных с неопределенными минимальными и максимальными значениями (например, рост или вес — кроме ситуаций, когда вы задаете конкретные границы). С другой стороны, тег `<meter>` можно использовать для отображения температуры, если интерпретировать его как шкалу термометра с заданным минимумом и максимумом.

В нашем примере шкала должна показывать, насколько близко мы подошли к цели $5000. Минимальное и максимальное значения известны, так что тег `<meter>` идеально подходит для этой задачи.

Простой прототип шкалы будет создан в новом документе HTML5. Чтобы продемонстрировать работу шкалы, в коде жестко зафиксировано текущее значение 2500.00.

html5_meter/index.html

```
<h3>Our Fundraising Goal</h3>
<meter title="USD" id="pledge_goal"
```

```
      value="2500.00" min="0" max="5000.00">
</meter>
<p>Help us reach our goal of $5000!</p>
```

Для управления шириной шкалы можно воспользоваться CSS. Создайте файл *stylesheets/style.css* и добавьте в него следующий код:

html5_meter/stylesheets/style.css

```
meter{
  width: 280px;
}
```

Не забудьте включить ссылку на файл CSS в секцию `<head>` страницы HTML:

html5_meter/index.html

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

Если посмотреть на результат в браузере, вы увидите вполне симпатичную шкалу. Однако теги `<meter>` поддерживаются не везде, так что нам понадобится хорошее обходное решение.

Обходное решение

Не все браузеры опознают тег `<meter>`. Однако проблема легко решается построением собственной шкалы с использованием jQuery и информации, встроенной в элемент `<meter>`. Для этого мы создадим новый файл *javascripts/fallback.js*, в котором будет храниться наше решение JavaScript. jQuery и этот файл следует загрузить в конце страницы.

Сначала, чтобы проверить поддержку элемента `<meter>` браузером, мы создаем элемент и проверяем, определен ли для него атрибут `max`. Если атрибут не определен, можно сделать вывод, что браузер не понял тег `<meter>` и не смог интерпретировать нашу разметку. Проверка реализуется функцией с именем `noMeterSupport()`:

html5_meter/javascripts/fallback.js

```
var noMeterSupport = function(){
  return(document.createElement('meter').max === undefined);
}
```

Затем мы используем jQuery для извлечения данных и построения шкалы при отсутствии поддержки `<meter>`.

Наша шкала будет состоять из внешнего прямоугольника `fakeMeter`, представляющего общую длину шкалы; внутреннего прямоугольника, который будет называться `fill`; и текстовой метки для вывода суммы в долларах. Также к элементу будут присоединены стили. После того как новый элемент будет готов, мы заменим тег `<meter>` собственной реализацией.

html5_meter/javascripts/fallback.js

```
if (noMeterSupport()) {
  var fakeMeter, fill, label, labelText, max, meter, value;
  meter = $("#pledge_goal");
  value = meter.attr("value");
  max = meter.attr("max");
  labelText = "$" + meter.val();

  fakeMeter = $("<div></div>");
  fakeMeter.addClass("meter");
  label = $("<span>" + labelText + "</span>");
  label.addClass("label");

  fill = $("<div></div>");
  fill.addClass("fill");
  fill.css("width",(value / max * 100) + "%");
  fill.append("<div style='clear:both;'><br></div>");
  fakeMeter.append(fill);
  fakeMeter.append(label);
  meter.replaceWith(fakeMeter);
}
```

Если в этом коде что-то покажется вам непонятным, в приложении Б приведен краткий обзор основных возможностей jQuery.

Когда код JavaScript будет готов, можно переходить к стилевому оформлению шкалы.

html5_meter/stylesheets/style.css

```
.meter{
  border: 1px solid #000;
  display: block;
  position: relative;
  width: 280px;
}
```

Мы назначаем рамку и ширину, а также выбираем режим относительного позиционирования (`relative`) для размещения метки внутри шкалы. Затем определяется внутреннее заполнение с градиентной заливкой:

html5_meter/stylesheets/style.css

```
.fill{
  background-color: #693;
  background-image: -webkit-gradient(
      linear,
      left bottom,
      left top,
      color-stop(0.37, rgb(14,242,30)),
      color-stop(0.69, rgb(41,255,57))
  );

  background-image: -moz-linear-gradient(
      center bottom,
      rgb(14,242,30) 37%,
      rgb(41,255,57) 69%
  );
}
```

Синтаксис градиентной заливки более подробно рассматривается в главе 8. После стилевого применения заливки необходимо разместить метку для суммы в долларах внутри прямоугольника.

html5_meter/stylesheets/style.css

```
.label{
  position: absolute;
  right: 0;
  top: 0;
  z-index: 1000;
}
```

Мы выполняем абсолютное позиционирование метки внутри прямоугольника. Метка располагается в слое над заливкой. Готовая шкала показана на рис. 4.

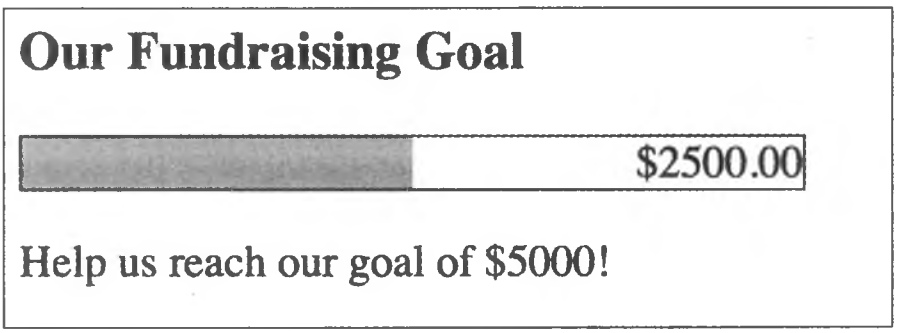


Рис. 4. Обходное решение для шкалы

Пользователи, отключившие JavaScript, увидят контент, находящийся в открывающем и закрывающем элементе `meter`; будьте внимательны с тем, какая информация в них будет размещена.

Индикатор прогресса

Если вам потребуется реализовать индикатор хода загрузки в веб-приложении, обратите внимание на тег `<progress>`, появившийся в HTML5.

Тег `<progress>` похож на `<meter>`, но он предназначен для активного отображения хода выполняемой операции — наподобие тех, которые отображаются при загрузке файлов. Шкала, напротив, предназначена для вывода статической информации — скажем, текущего объема пространства на сервере, доступного для некоторого пользователя. Разметка индикатора прогресса очень похожа на разметку шкалы:

html5_meter/progress.html

```
<progress id="progressbar" min="0" max="100" value="0"></
progress>
```

Тег `<progress>`, как и `<meter>`, пока поддерживается не всеми браузерами. Чтобы справиться с проблемой, используйте JavaScript для извлечения значений из индикатора и построения собственной визуализации. Также можно воспользоваться готовым компонентом HTML5-Progress[1], созданным Леа Веру (Lea Verou).

[1] http://lea.verou.me/polyfills/progress/

Рецепт 3. Создание всплывающих окон с пользовательскими атрибутами данных

Каждый, кто в своих веб-приложениях использовал JavaScript для получения информации из документа, знает, что решение этой задачи требует немалых ухищрений. Разработчику приходится вставлять дополнительную информацию в обработчики событий или же злоупотреблять атрибутами `rel` или `class`. С появлением пользовательских атрибутов данных все трудности уходят в прошлое.

Пользовательские атрибуты данных начинаются с префикса `data-`; они игнорируются валидаторами документов HTML5. Пользовательский атрибут данных можно присоединить к любому элементу, будь то метаданные о фотографии, широта и долгота точки на карте или, как в примере настоящего раздела, размеры всплывающего окна. А самое замечательное — пользовательские атрибуты данных можно использовать прямо сейчас практически в любом браузере, потому что их значения легко читаются кодом JavaScript.

Отделение поведения от контента, или Чем плохо решение с onclick

За последние годы всплывающие окна приобрели плохую репутацию, и вполне заслуженно. Они часто используются для того, чтобы заставить вас просмотреть рекламу, спровоцировать неосторожного пользователя на установку вируса или шпионской программы или, еще хуже, сообщить личную информацию, которая затем перепродается злоумышленником. Неудивительно, что в большинстве браузеров имеются механизмы блокировки всплывающих окон.

Впрочем, всплывающие окна не всегда плохи. Разработчики веб-приложений часто используют их для вывода справки, дополнительных настроек или других важных аспектов пользовательского интерфейса. Чтобы всплывающие окна не вызывали раздражения у пользователей, их следует реализовать ненавязчиво. На странице отдела кадров AwesomeCo находятся ссылки, которые выводят справочную информацию во всплывающих окнах. Большинство из них выглядит так:

html5_popups_with_custom_data/original_example_1.html

```
<a href='#'
  onclick="window.open('help/holiday_pay.html',WinName,
                       'width=300,height=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, о которых 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.

html5_popups_with_custom_data/original_example_2.html

```
<a href='holiday_pay.html'
  onclick="window.open(this.href,WinName,'width=300,heig
ht=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. Написать интерактивную часть JavaScript на таком фундаменте может быть намного проще.

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 размер итогового кода HTML получается намного больше необходимого.

Вместо этого назначьте каждому якорю на странице идентифицирующий класс.

html5_popups_with_custom_data/original_example_3.html

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

Чтобы обработка событий не создавала проблем совместимости, мы воспользуемся jQuery. В конце страницы, непосредственно перед закрывающим тегом `<body>`, подключите библиотеку jQuery.

html5_popups_with_custom_data/original_example_3.html

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

Затем прямо под этой строкой добавьте новый тег `<script>` со следующим кодом:

html5_popups_with_custom_data/original_example_3.html

```
$("a.popup").click(function(event){
  event.preventDefault();
  window.open(this.getAttribute('href'));
});
```

Мы используем селектор jQuery для получения элемента с классом `popup`, после чего добавляем наблюдателя для события `click` для каждого элемента.

Функция, передаваемая 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

```
$("a.popup").click(function(event){
  event.preventDefault();
  var link = this;
  var href = link.getAttribute("href");
  var height = link.getAttribute("data-height");
  var width = link.getAttribute("data-width");

  window.open (href,"popup",
    "height=" + height +",width=" + width + "");
});
```

jQuery используется исключительно для обработки события `click()`. Элемент, на котором был сделан щелчок, представлен в функции-обработчике ключевым словом `this`. Используя `getAttribute()`, мы получаем все необходимые атрибуты из элементов для создания всплывающего окна.

Вот и все! Ссылка открывается в новом окне.

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.

К пользовательским атрибутам данных также можно обратиться другим способом — через dataset. При этом пользовательские атрибуты данных преобразуются в свойства, к которым можно обращаться следующим образом:

html5_popups_with_custom_data/popup_dataset.html

```
var height = link.dataset.height;
var width = link.dataset.width;
```

Это удобно, но вы должны помнить о паре возможных проблем. Во-первых, это решение не работает в Internet Explorer 10 и более ранних версий, так что универсальным его никак не назовешь. Во-вторых, если у вас имеется пользовательский атрибут данных вида data-mobile-image-size, к нему придется обращаться в формате dataset.mobileImageSize. Имена свойств dataset строятся по схеме чередования регистра символов (camel case).

Предупреждение

В этом примере для передачи дополнительной информации сценарию клиентской стороны использовались пользовательские атрибуты данных. 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.

Рецепт 4. Применение списков описаний для определения FAQ

Если на сайтах, управляемых контентом, и есть что-то постоянное, так это списки часто задаваемых вопросов, или FAQ (Frequently Asked Questions). Хорошие сайты заполняют FAQ вопросами, которые действительно интересуют людей; остальные используют этот раздел для ответов, которые уже должен был дать основной раздел сайта. Впрочем, независимо от того, какую информацию содержат списки FAQ, традиционно у разработчиков возникают проблемы с выбором разметки для их реализации.

Веб-разработчики применяли самые разные варианты, от упорядоченных списков до тегов `<div>` с классами и стилевым оформлением, но такие решения оставляют желать лучшего с точки зрения семантики. Нам нужна возможность связать вопрос с ответом, а это можно легко и семантически правильно сделать при помощи элемента `<dl>`.

В предыдущих версиях HTML элемент `<dl>` назывался *списком определений* (Definition List) и предназначался для связывания термина с определением. В HTML5 элемент `<dl>` превратился в *список описаний* (Description List). Хотя код остался почти тем же, изменение в спецификации более ясно показывает гибкость возможных применений этого элемента.

И это очень хорошо, потому что компании AwesomeCo понадобился список FAQ, объясняющий, чем занимается эта компания.

Структура

Структура списка FAQ весьма проста. Мы используем тег `<dl>` для определения самого списка, а каждый вопрос будет представлен тегом `<dt>`. Ответы на вопросы будут размещаться в элементах `<dd>`.

```
<article>
  <h1>AwesomeCo FAQ</h1>
  <dl>
    <dt>What is it that AwesomeCo actually does?</dt>
    <dd>
      <p>
        AwesomeCo creates innovative solutions for business that
        leverage growth and promote synergy, resulting in a better
        life for the global community.
      </p>
```

```
    </dd>
  </dl>
</article>
```

Стилевое оформление по умолчанию достаточно хорошо работает в большинстве браузеров, как видно из рис. 5; ответы выводятся под вопросами с отступом.

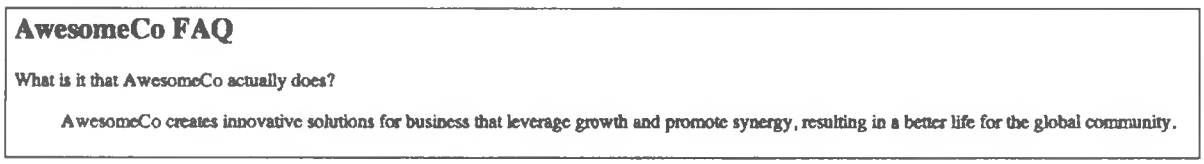


Рис. 5. Список FAQ без применения стилей

После создания такой разметки мы можем применять стили без добавления классов в элементы HTML. Также достаточно легко свернуть отдельные элементы FAQ средствами JavaScript, если потребуется сэкономить экранное пространство на смартфонах.

Тег `<dl>` поддерживается всеми браузерами; в HTML5 изменился только рекомендуемый способ его использования с контентом. Обходное решение на этот раз не понадобится.

Перспективы

Новые теги и атрибуты открывают немало интересных возможностей. Например, вы можете 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. Загляните в спецификацию; вы найдете в ней немало элементов, которые со временем будут реализованы в браузерах, включая разметку диалоговых окон (`<dialog>`), выделение текста (`<mark>`) и многое другое.

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`. Или, допустим, их можно использовать для вывода 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!

Новые возможности веб-форм

3

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 jQuery UI[1] или реализуете собственные элементы управления 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 управления, добавленные на страницу, не конфликтовали с другими элементами управления или библиотеками JavaScript на странице. Вы можете потратить несколько часов на собственную реализацию календаря, а потом обнаружить, что она не работает, потому что плагин был написан недостаточно аккуратно и несовместим с последней версией jQuery, используемой в проекте.

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. Также будут рассмотрены простые средства проверки данных на стороне клиента. В за-

[1] http://jqueryui.com/

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

Конкретно в этой главе рассматриваются следующие возможности:

`<input type="email">`
: Поле для ввода адреса электронной почты. [O10.1, IOS, A3]

`<input type="url">`
: Поле для ввода URL-адреса. [O10.1, IOS5, A3]

`<input type="range">`
: Ползунок. [C5, S4, F23, IE10, O10.1]

`<input type="number">`
: Поле для ввода числовых данных, часто в виде элемента-счетчика. [C5, S5, O10.1, IOS5, A3]

`<input type="color">`
: Поле для выбора цвета. [C5, O11]

`<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`. [S5, O10.1]

`<input type="search">`
: Поле для ввода ключевых слов поиска. [C5, S4, O10.1, IOS]

`<input type="text" autofocus>`
: Поддержка передачи фокуса конкретному элементу формы. [C5, S4]

`<input type="email" placeholder="me@example.com">`
: Поддержка автоматического включения текста в поле формы. [C5, F4, S4]

`<input type="email" required>`
: Блокировка отправки страницы, если поле осталось незаполненным. [C23, F16, IE10, O12]

`<input pattern="/^(\s*|\d+)$/">`
: Блокировка отправки страницы, если содержимое поля не соответствует заданному образцу. [C23, F16, IE10, O12]

`<p contenteditable>lorem ipsum</p>`
: Поддержка редактирования контента «на месте» в браузере. [C4, F3.5, S3.2, IE6, O10.1, iOS5, A3]

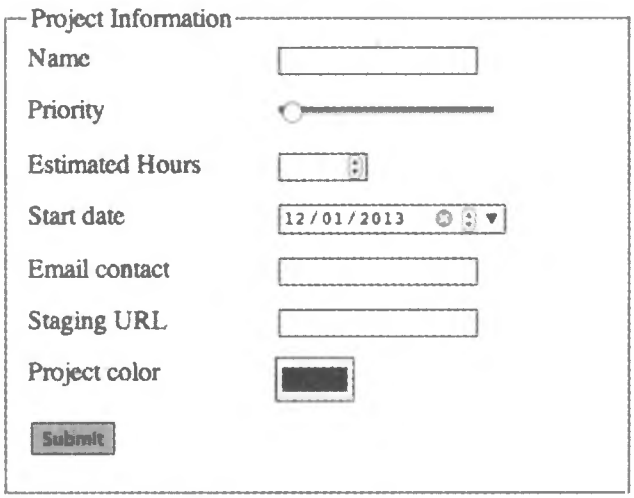
Итак, для начала познакомимся поближе с некоторыми из этих исключительно полезных типов полей.

Рецепт 5. 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, браузер Safari для iOS при вводе данных в полях `URL` и `email` отображает альтернативную раскладку с удобным вводом специальных символов `@`, `.`, `:` и `/`.

Компания 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. В конечном итоге форма должна выглядеть примерно так.



Описание формы

Давайте создадим новую страницу HTML5 с базовой формой HTML, которая осуществляет отправку данных методом POST. Предполагается, что где-то существует страница, которая эту форму обрабатывает. 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" 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. После того как форма будет построена, мы воспользуемся правилами CSS для упорядочения ее элементов.

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`.

html5_forms/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, Safari 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. Помимо упрощения ввода данных, использование этого поля четко описывает тип содержащихся в нем данных.

html5_forms/index.html

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

В Chrome, Safari и 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, так что ввод придется ограничить при помощи сценариев или средствами проверки данных HTML5, которые будут рассматриваться в рецепте 8.

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

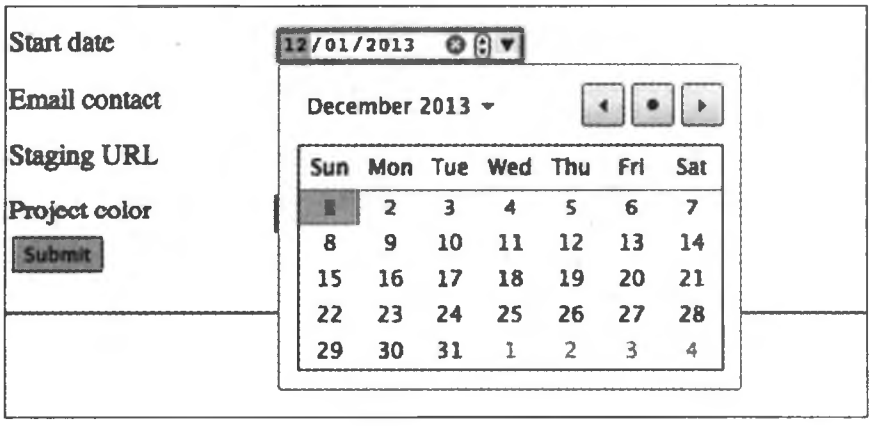
Дата

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

html5_forms/index.html

```
<label for="start_date">Start date</label>
<input type="date" name="start_date" id="start_date"
    value="2013-12-01">
```

Поле `date` четко описывает тип содержащихся в нем данных, а при правильной реализации пользователь увидит календарный виджет наподобие того, который отображается в Chrome.



На момент написания книги среди браузеров для настольных систем полноценный календарный элемент для выбора даты поддерживался только в Chrome и Opera, но некоторые мобильные браузеры используют это поле для упрощенного ввода месяца, даты и года. Все остальные браузеры просто отображают текстовое поле.

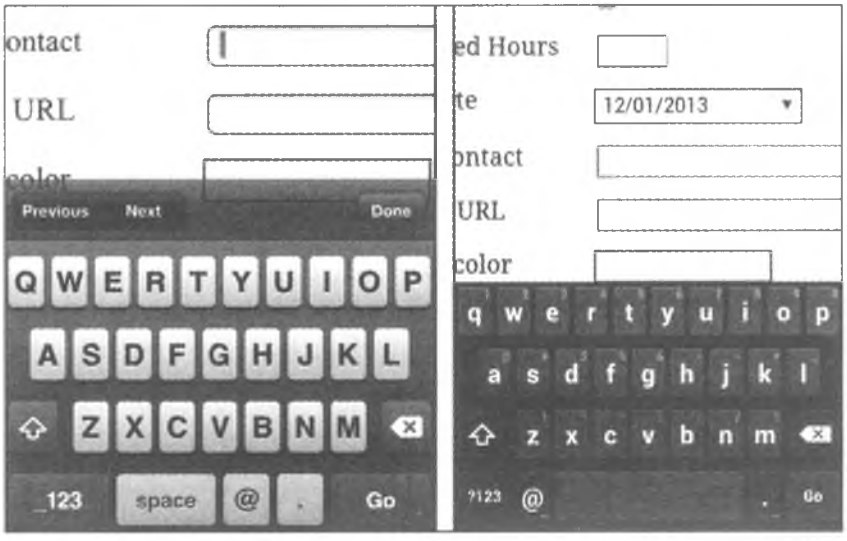
Адрес электронной почты

Поле типа `email` предназначено для хранения либо отдельного адреса, либо списка адресов электронной почты. Таким образом, это поле идеально подходит для нашего приложения.

html5_forms/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. Обратите внимание на присутствие символа @ в клавиатурах iPhone и Android.



URL-адрес

На форме из нашего примера должен размещаться URL-адрес для просмотра предварительной версии проекта. Этот тип поля также адаптирован для ввода URL-адресов. 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">
```

Этот тип также особенно удобен, если ваши посетители просматривают сайт с устройств iPhone или Android — 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 мобильного браузера).

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 лишь немногие браузеры отображали 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.

Применение стилей к форме

Мы применим к форме простейшее оформление CSS. Создайте новый файл с именем *stylesheets/style.css* и включите ссылку на него в секцию `<head>` страницы формы.

html5_forms/index.html

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

Для начала уберем нумерацию, поля и отступы из списка:

html5_forms/stylesheets/style.css

```
ol{
  list-style: none;
  margin: 0;
  padding :0;
}
ol li{
  clear: both;
  margin: 0 0 10px 0;
  padding: 0;
}
```

Затем выровняем метки и поля ввода, а также слегка изменим оформление полей ввода.

html5_forms/stylesheets/style.css

```
label{
  float: left;
  width: 150px;
}

input{ border: 1px solid #333; }

input:focus{ background-color: #ffe; }
```

Псевдокласс :focus позволяет применить стилевое оформление только к тому полю, которое в настоящий момент имеет фокус ввода.

Мы построили вполне презентабельную и семантически правильную веб-форму. Можно пойти еще дальше и идентифицировать каждый из типов полей средствами CSS.

Chrome и Opera поддерживают большинство новых элементов, но при открытии страницы в Firefox, Safari или Internet Explorer многие поля отображаются как обычные текстовые поля.

Обходное решение

Браузеры, в которых новые типы полей не реализованы, просто заменяют их текстовыми полями, чтобы с формой по крайней мере можно было работать. На этой стадии можно принять решение об использовании другого виджета или библиотеки. Например, вы проверяете, поддерживает ли браузер элемент календаря, и если не поддерживает — добавляете его средствами jQuery UI. Со временем полноценные элементы будут поддерживаться всеми основными браузерами, и тогда «заплатки» можно будет удалить. Процесс выглядит практически одинаково для всех элементов управления, для которых необходимо создать обходное решение.

Замена элемента выбора цвета

Элемент выбора цвета легко идентифицируется и заменяется при помощи jQuery и плагина jQuery-simple-color[1]. Мы ищем любое поле <input> с типом color и применяем плагин jQuery:

```
$('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.

[1] http://recursive-design.com/projects/jquery-simple-color/

Обычно при использовании таких плагинов, как SimpleColor, мы загружаем плагин, присоединяем его к странице в теге `<script>`, а затем присоединяем его к элементу при помощи отдельного фрагмента JavaScript. Но 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`. Если поле не поддерживается, плагин загружается динамически.

Для начала, поскольку нам понадобится библиотека jQuery, мы загружаем ее непосредственно перед закрывающим тегом `<body>`:

html5_forms/index.html

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

Затем создается новый файл с именем *javascripts/fallbacks.js*, который включается в конец страницы HTML тегом `<script>`;

html5_forms/index.html

```
<script src="javascripts/fallbacks.js"></script>
```

В файле *javascripts/fallbacks.js* создается функция, проверяющая поддержку `color` в браузере:

html5_forms/javascripts/fallbacks.js

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

Сначала мы 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.

В строке 6 начинается самое интересное. В некоторых браузерах 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.

Наконец, мы вызываем функцию обнаружения. Чтобы плагин не загружался тогда, когда он не нужен, мы используем JavaScript для внедрения нового тега `<script>`, который загружает плагин и активизирует его после загрузки сценария.

html5_forms/javascripts/fallbacks.js

```
var applyColorPicker = function(){
  $('input[type=color]').simpleColor();
};

if (!hasColorSupport()){
  var script = document.createElement('script');
  script.src = "javascripts/jquery.simple-color.js";

  if(script.readyState){ // Поддержка IE
    script.onreadystatechange = function () {
      if (this.readyState === 'loaded' ||
          this.readyState === 'complete'){
        script.onreadystatechange = null;
        applyColorPicker();
      }
    };
  }else{
    // Другие браузеры
    script.onload = applyColorPicker;
  }

  document.getElementsByTagName("head")[0].appendChild(script);
}
```

Internet Explorer заставляет нас использовать `onreadystatechange` и проверять значение свойства `readystate`. Другие браузеры просто позволяют присоединить имя функции к событию `onload`. Включение обоих вариантов гарантирует, что обходное решение будет работать в любом случае.

Теперь решение работает во всех браузерах. Осталось добавить немного дополнительного кода CSS, чтобы элемент выбора цвета выравнивался с другими элементами:

html5_forms/stylesheets/style.css

```
  .simpleColorContainer, .simpleColorDisplay{
  float: left;
}
```

Процесс создания обходных решений для других полей формы выглядит аналогично — мы проверяем, имеется ли в браузере встроенная поддержка элемента, и если нет — загружаем необходимую библиотеку и применяем версию JavaScript. В будущем надобность в элементах JavaScript отпадет, и мы сможем полагаться исключительно на поддержку браузера. Это решение работает, но его надежность оставляет желать лучшего: оно работает только с конкретными браузерами и предназначено только для элемента выбора цвета. 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.

Библиотека Modernizr загружается в секции `<head>` документа после ссылок CSS. Она предоставляет объект `Modernizr`, который может использоваться для проверки функциональности и загрузки обходных решений. Например, вот как это делается для нашего обходного решения с `color`:

```
if(Modernizr.inputtypes.color){
  // Есть встроенная поддержка color
}else{
  // Нет поддержки color
}
```

При отсутствии встроенной поддержки приходится загружать дополнительные библиотеки. Мы можем воспользоваться методом Modernizr `load()` для проверки функциональности и загрузки других сценариев, а затем выполнить свой код после загрузки кода. Это выглядит примерно так:

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

html5_forms/modernizr/javascripts/fallbacks.js

```
var applyColorPicker = function(){
  $('input[type=color]').simpleColor();
};

Modernizr.load(
  {
    test: Modernizr.inputtypes.color,
    nope: "javascripts/jquery.simple-color.js",
    callback: function(url, result){
      if (!result){
        applyColorPicker();
      }
    }
  }
);
```

Метод `load()` получает объект JavaScript, который определяет конкретную проверку функциональности и действия, выполняемые при ее наличии или отсутствии. Мы используем `yep` для загрузки сценариев в случае поддержки функциональности и `nope` для загрузки сценариев при ее отсутствии. Также можно определить функцию обратного вызова, которая выполняется при завершении загрузки внешнего файла. В нашем примере проверяется поддержка элемента `color`; если он недоступен, мы загружаем плагин jQuery. Затем в функции обратного вызова используется переменная `result`, равная `true`, если браузер поддерживает элемент выбора цвета, или `false` в противном случае. Другими словами, `result` содержит результат выполненной проверки.

Формально проверять `result` не обязательно, потому что функция обратного вызова запускается только при загрузке сценария. В нашем примере сценарии загружаются только при отсутствии встроенной поддержки. Однако мы также могли использовать вариант `yep`, позволяющий загрузить дополнительные сценарии при поддержке поля `color` браузером. Эти сценарии также активизируют функцию обратного вызова, и тогда в этой функции придется проверять причину срабатывания проверкой `result`.

Функция `load()` построена на базе библиотеки *yepnope.js*. Чтобы понять, как она работает, обращайтесь к документации *yepnope.js*[1].

[1] http://yepnopejs.com/

Чтобы использовать метод `load()`, нам пришлось построить собственную версию Modernizr с использованием сетевых инструментов[1]. Построитель включает библиотеку Yepnope. Я включил полную версию Modernizr в архив примеров кода, но такое решение вряд ли годится для реального приложения — в проект следует включать только действительно необходимые компоненты Modernizr.

Прежде чем включать Modernizr в свои проекты, обязательно разберитесь в том, как работает исходный код библиотеки. 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. Например, библиотека Modernizr не справлялась с частичной поддержкой элемента выбора цвета в Safari, поэтому разработчикам пришлось быстро создавать «заплатки» на своих сайтах в ожидании обновления Modenizr. 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!

Из-за сложностей, связанных с проверкой функциональности в разных браузерах, мы будем использовать Modernizr везде, где это имеет смысл.

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

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

Рецепт 6. 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.

html5_forms/index.html

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

Конструкции вида `autofocus="true"` или `autofocus="autofocus"` не нужны. Если атрибут `autofocus` присутствует, то браузер применяет его.

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 и передайте фокус элементу. Вероятно, это самое простое обходное решение из всех приведенных в книге. Включите следующий фрагмент в файл *javascripts/fallbacks.js*:

html5_forms/autofocus/javascripts/fallbacks.js

```
if (!Modernizr.autofocus){
  $('input[autofocus]').focus();
}
```

Для передачи фокуса используется jQuery. То же самое можно было сделать на уровне «простого» JavaScript, но мы уже загрузили jQuery и можем использовать для обнаружения поля с автофокусом селекторы атрибутов (вместо включения в поле специального класса или идентификатора).

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 7. Заполняющий текст

Заполняющий текст сообщает пользователям, как следует вводить данные в поле формы. Он не является заменой тега `<label>`, а всего лишь предоставляет подсказку для пользователя.

На сайте поддержки 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 будет включаться и в другие поля.

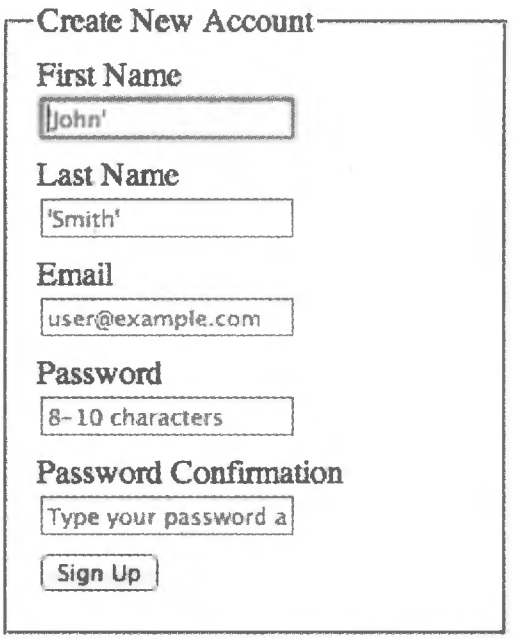


Рис. 6. Заполняющий текст подсказывает, какую информацию следует ввести в поле

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

html5_placeholder/index.html

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

После добавления заполняющего текста во все поля разметка формы выглядит примерно так.

html5_placeholder/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.

html5_placeholder/stylesheets/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

Предложено много решений, реализующих функциональность заполнителей в старых браузерах; одно из простейших основано на использовании плагина jQuery Placeholder[1]. Плагин используется по тем же принципам, по которым мы реализовали обходное решение для поддержки `color` — мы проверяем поддержку заполнителей и загружаем и активизируем плагин только при ее отсутствии. Создайте новый файл с именем *javascripts/fallbacks.js*, в котором будет храниться код проверки и обходного решения. Подключите файл к странице HTML при помощи тега `<script>` в конце страницы:

[1] https://github.com/mathiasbynens/jquery-placeholder

html5_placeholder/index.html

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

Мы используем Modernizr точно так же, как это было сделано на с. 70; это означает, что библиотека Modernizr включается в секцию `<head>` страницы:

html5_placeholder/index.html

```
<script src='javascripts/modernizr.js'></script>
```

В файле *javascripts/fallbacks.js* объявляется функция, активизирующая плагин jQuery Placeholder:

html5_placeholder/javascripts/fallbacks.js

```
var applyPlaceholders = function(){
  $("input").placeholder();
}
```

Затем мы используем Modernizr для проверки поддержки заполняющего текста, загрузки плагина и активизации метода обратного вызова:

html5_placeholder/javascripts/fallbacks.js

```
Modernizr.load(
  {
    test: Modernizr.placeholder,
    nope: "javascripts/jquery.placeholder.js",
    callback: function(url, result){
      if (!result){
        applyPlaceholders();
      }
    }
  }
);
```

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

Заполнители подсказывают пользователю, какого рода данные должны вводиться в поле. Но иногда этого оказывается недостаточно — требуется убедиться в том, что пользователь правильно заполнил поля формы.

Рецепт 8. Проверка пользовательского ввода без использования JavaScript

Создавая веб-форму, мы стремимся к тому, чтобы получить данные от пользователя для сохранения или преобразования в полезную информацию. Одни данные могут быть обязательными, другие можно опустить. Иногда данные должны вводиться в строго определенном формате и их нужно проверить перед сохранением, чтобы у пользователя была возможность исправить ошибки. Традиционно разработчики проверяют пользовательский ввод в серверном коде, потому что средства проверки данных JavaScript на стороне клиента легко отключаются. Проверка данных на стороне сервера снижает эффективность работы пользователя — чтобы узнать о допущенной ошибке, пользователь должен заполнить поля, отправить форму и дождаться обновления всей страницы. Чтобы ускорить обратную связь с пользователем, разработчики все равно пишут проверку данных на стороне клиента на JavaScript. Фактически проверка реализуется дважды, что создает свои проблемы.

В HTML5 появилась пара атрибутов элементов форм, которые могут использоваться для проверки данных на стороне клиента. Это позволяет обнаружить простые ошибки ввода до того, как пользователь отправит запрос серверу без написания кода JavaScript.

Страница регистрации пользователя на сайте поддержки AwesomeCo отлично подходит для тестирования новых атрибутов. Пользователь должен ввести свое имя, фамилию и адрес электронной почты. Также необходимо ввести пароль. Начнем с проверки заполнения полей имени, фамилии и электронной почты.

Чтобы объявить поле формы обязательным для заполнения, следует добавить в соответствующий элемент атрибут `required` (по аналогии с тем, как это делалось с атрибутом `autofocus`). Таким образом, атрибут `required` должен присутствовать в полях имени, фамилии и электронной почты исходной формы.

html5_validation/index.html

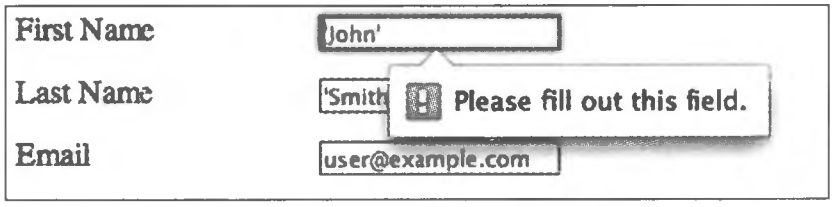
```
<li>
  <label for="first_name">First Name</label>

  <input id="first_name" type="text"
    autofocus="true"
    required
```

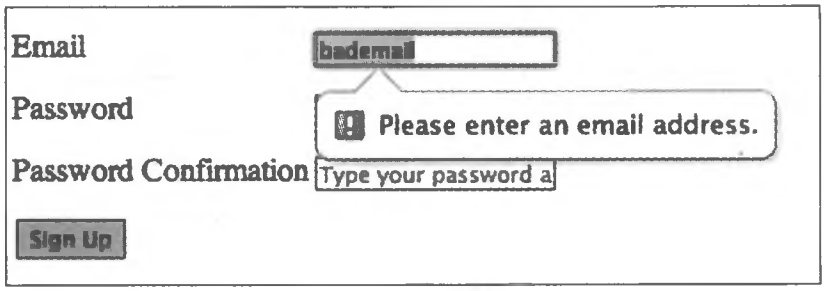
```
    name="first_name" placeholder="'John'">
</li>
<li>
  <label for="last_name">Last Name</label>
  <input id="last_name" type="text"
    required
    name="last_name" placeholder="'Smith'">
</li>
<li>
  <label for="email">Email</label>

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

Браузеры, поддерживающие этот атрибут, блокируют отправку формы, если обязательные поля не были заполнены. Пользователь получает сообщение об ошибке, а нам не приходится писать ни одной строки проверочного кода JavaScript. Вот как это выглядит в Chrome:



Поле электронной почты также назначено обязательным для заполнения, при этом мы получаем дополнительное преимущество: пользователь должен ввести значение, по крайней мере похожее на адрес электронной почты.



Это еще одна убедительная причина для использования полей формы, описывающих вводимые данные.

С первыми тремя обязательными полями мы разобрались, но теперь необходимо проследить за тем, чтобы поле пароля содержало не менее восьми символов.

Проверка по регулярным выражениям

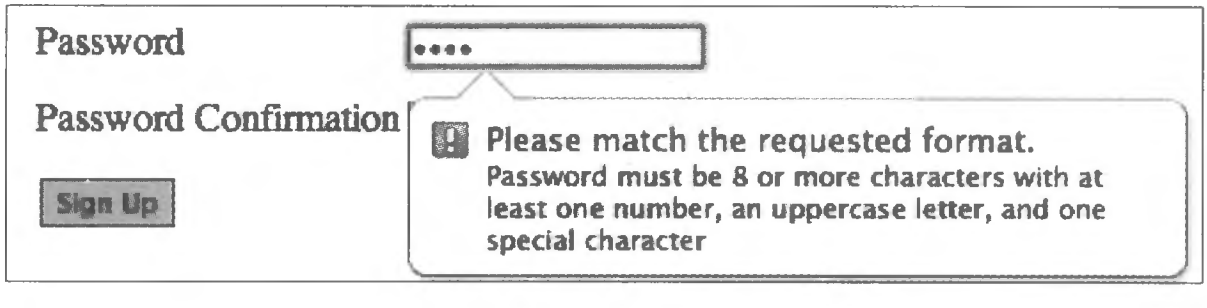
Атрибут `pattern` позволяет задать регулярное выражение, по которому будут проверяться данные пользователя. Браузер уже знает, как проверять адреса электронной почты и URL-адреса, но для поля с паролем устанавливаются особые правила. Для этого в поле Password включается атрибут `pattern` с регулярным выражением, требующим, чтобы поле содержало не менее 8 символов, среди которых присутствуют хотя бы одна цифра, буква верхнего регистра и один специальный символ.

html5_validation/index.html

```
<li>
  <label for="password">Password</label>
  <input id="password" type="password" name="password" value=""
    autocomplete="off" placeholder="8-10 characters"
    pattern="^(?=.{8,})(?=.*[a-z])(?=.*[A-Z])(?=.*[\d])
            (?=.*[\W]).*$"
    title="Password must be 8 or more characters with at
        least one number, an uppercase letter, and one special
        character"
  />
</li>
```

Обратите внимание на использование атрибута `title` для определения содержательного описания данных, вводимых пользователем.

Теперь при попытке ввести данные, не соответствующие регулярному выражению, выводится сообщение об ошибке, в которое включается содержимое атрибута `title`.



Тот же шаблон используется и в поле подтверждения пароля.

html5_validation/index.html

```
<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"
    pattern="^(?=.{8,})(?=.*[a-z])(?=.*[A-Z])(?=.*[\d])(?=.*[\W]).*$"
    title="Password confirmation must be 8 or more characters with at
        least one number, an uppercase letter, and one special
        character"
  />
</li>
```

К сожалению, атрибуты проверки данных не позволяют убедиться в том, что пароль совпадает с подтверждением. Эту задачу придется решать средствами JavaScript.

Стилевое оформление полей

У нашей формы уже имеются базовые стили, но теперь у нас появилась дополнительная информация о полях, при помощи которой можно более наглядно сообщить об обнаруженных ошибках. Некоторые браузеры (такие, как Firefox) выделяют недействительные поля при потере ими фокуса. Другие браузеры не предоставляют обратной связи до момента отправки формы. Небольшой фрагмент CSS позволит нам предоставить мгновенную обратную связь пользователю.

Для этого в объявлениях стилей будут использоваться псевдоклассы `:valid` и `:invalid`:

html5_validation/stylesheets/style.css

```
input[required]:invalid, input[pattern]:invalid{
  border-color: #A5340B;
}

input[required]:valid, input[pattern]:valid{
  border-color: #0B9900;
}
```

Теперь при изменении содержимого полей пользователь будет получать визуальную обратную связь.

Обходное решение

Простейшее обходное решение — вообще ничего не делать. Браузеры будут игнорировать неподдерживаемые атрибуты `required` и `patter`, а поиском ошибок будет заниматься код на стороне сервера. Но если вы сочтете, что этого недостаточно, используйте значения атрибутов `required` и `pattern` для построения собственных средств проверки или же воспользуйтесь библиотекой — например, чрезвычайно мощной библиотекой H5F[1]. В отличие от других библиотек, при использовании H5F не нужно проверять версию браузера, потому что она делает это автоматически и задействует всю существующую поддержку в браузере.

Чтобы использовать эту библиотеку, необходимо включить ее в страницу и активизировать. Для проверки функциональности и загрузки библиотеки мы снова воспользуемся Modernizr. Хотя H5F выполняет проверку самостоятельно, мы хотим свести к минимуму загрузку внешних сценариев, а Modernizr уже используется для обходного решения, обеспечивающего поддержку заполняющего текста. Существующая функция `load()` должна выглядеть так:

html5_validation/javascripts/fallbacks.js

```
Modernizr.load([
  {
    test: Modernizr.placeholder,
    nope: "javascripts/jquery.placeholder.js",
    callback: function(url, result){
      if (!result){
        applyPlaceholders();
      }
    }
  }

  ,
  {
    test: Modernizr.pattern && Modernizr.required,
    nope: "javascripts/h5f.min.js",
    callback: function(url, result){
      if (!result) {
        configureH5F();
      }
    }
  }
]);
```

[1] https://github.com/ryanseddon/H5F

Для активизации H5F пишется следующий код:

html5_validation/javascripts/fallbacks.js

```
  var configureH5F = function(){
  H5F.setup(document.getElementById("create_account"));
};
```

Функция `load()` получает строку, объект или массив объектов. На этот раз `Modernizr .load()` должны передаваться два объекта вместо одного. Для этого оба объекта необходимо поместить в массив (поэтому объекты заключаются в квадратные скобки).

Мы используем `document.getElementById()`, потому что H5F требуется обычный элемент DOM. Если бы мы воспользовались jQuery для выборки элемента, то получили бы объект jQuery вместо объекта Element, и библиотека H5F не знала бы, что с ним делать.

Проверка на стороне клиента сообщает пользователям о допущенной ошибке без ожидания ответа сервера или обновления страницы. Но помните, что эта функция может быть отключена или просто неправильно реализована, так что вам все равно придется предусмотреть стратегию проверки данных на стороне сервера.

Поля форм — не единственный способ ввода данных на веб-странице. Давайте посмотрим, как разрешить пользователю вводить текст прямо в обычных элементах HTML.

Рецепт 9. 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 атрибут `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. 7.

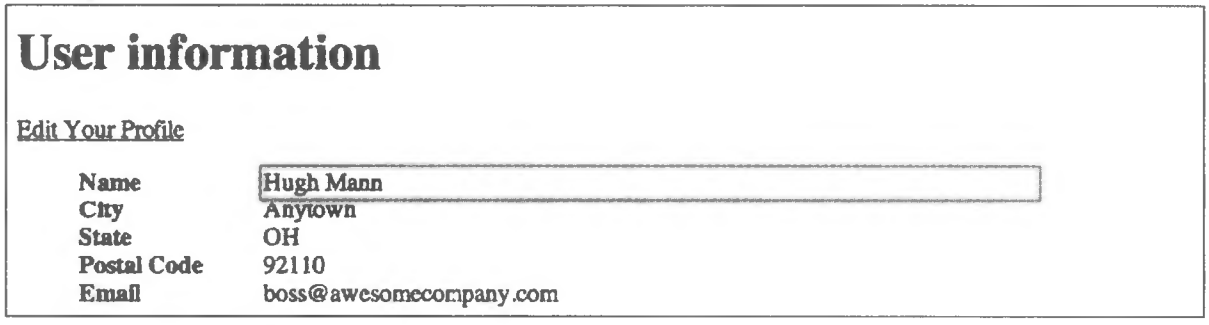


Рис. 7. Простое редактирование «на месте»

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. Там, где это возможно, стройте решения с JavaScript на основе

решений, не использующих JavaScript. В долгосрочной перспективе это будет способствовать улучшению разметки, кода и доступности решения.

Форма профиля

В HTML5 появился атрибут `contenteditable`, который поддерживается практически всеми элементами. Простое добавление этого атрибута превращает элемент в поле с возможностью редактирования. Итак, построим форму с профилем пользователя. Создайте новую страницу HTML с именем *show.html*:

html5_content_editable/show.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>Show User</title>
    <link rel="stylesheet" href="stylesheets/style.css">
  </head>
  <body id="forms">
  </body>
</html>
```

В тег `<body>` включаются редактируемые поля.

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. Кроме базового оформления для выравнивания полей, мы воспользуемся селекторами CSS3 для идентификации редактируемых полей, чтобы они изменяли цвет при наведении указателя мыши или выделении. Создайте новый файл с именем *stylesheets/style.css*, содержащий следующий код:

html5_content_editable/stylesheets/style.css

```
1   ul{list-style:none;} Line 1
-
-   li > b, li > span{
-     display: block;
5     float: left; 5
-     width: 100px;
-   }
-
-   li > span{
10    width:500px; 10
-     margin-left: 20px;
-   }
-
-   li > span[contenteditable]:hover{
15    background-color: #ffc; 15
-   }
-
-   li > span[contenteditable]:focus{
-     background-color: #ffa;
20    border: 1px shaded #000; 20
-   }
-
-   li{clear:left;}
```

В строке 3 мы выравниваем метку и `<span>`. Затем в строках 14 и 18 добавляются эффекты `hover` и `focus` при помощи селекторов атрибутов CSS.

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, то ничего нового в происходящем здесь вы не увидите.

Мы создаем новый файл с именем *javascripts/edit.js* и подключаем его и библиотеку jQuery в конце страницы HTML, прямо перед закрывающим тегом `<body>`:

html5_content_editable/show.html

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

Теперь можно написать код сохранения данных при изменении:

html5_content_editable/show.html

```
$("#edit_profile_link").hide();
var status = $("#status");
$("span[contenteditable]").blur(function(){
  var field = $(this).attr("id");
  var value = $(this).text();
  var resourceURL = $(this).closest("ul").attr("data-url");

  $.ajax({
    url: resourceURL,
    dataType: "json",
    method: "PUT",
    data: field + "=" + value,
    success: function(data){
      status.html("The record was saved.");
    },
    error: function(data){
      status.html("The record failed to save.");
```

```
        }
      });
    });
}
```

Мы добавляем слушателя событий для каждого элемента `span` на странице, у которого атрибут `contenteditable` равен `true`. Когда пользователь переходит к другому полю, нам остается лишь отправить данные сценарию на стороне сервера с использованием метода jQuery `ajax()`. При программировании веб-страницы URL-адрес на стороне сервера был включен в атрибут `data-url` тега `<ul>`, поэтому мы извлекаем URL-адрес и строим запрос к серверу. Это всего лишь пример — у нас нет подсистемы для сохранения информации, а ее написание выходит за рамки книги. Тем не менее некоторые способы сохранения данных пользователя на клиентской машине рассматриваются в главе 9.

Обходное решение

Некоторые из использованных нами приемов будут работать не у всех пользователей. Прежде всего, появилась нежелательная зависимость от JavaScript для сохранения отредактированных результатов на сервере. 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:

- Браузер пользователя не поддерживает `contenteditable`.
- Пользователь работает в современном браузере, но отключает JavaScript просто из-за своей неприязни к JavaScript (это происходит сплошь и рядом... намного чаще, чем можно ожидать).
- Пользователь работает за брандмауэром, который отфильтровывает JavaScript. Хотите — верьте, хотите — нет, но такие брандмауэры существуют и усложняют жизнь как пользователей, так и разработчиков.

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 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>
```

Мы добавим в *stylesheets/style.css* стили, улучшающие внешний вид формы. Эти стили похожи на те, которые уже использовались нами для других форм:

html5_content_editable/stylesheets/style.css

```
ol{
  padding :0;
  margin: 0;
  list-style: none;
}

ol > li{
  padding: 0;
  clear: both;
  margin: 0 0 10px 0;
}

label{
  width: 150px;
  float: left;
}
/* EN:edit_styles */
```

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 должна включаться только при поддержке редактирования контента. Проверка выполняется очень просто; Modernizr для этого не понадобится. Нужно лишь проверить наличие у элемента атрибута `contenteditable`.

html5_content_editable/javascripts/edit.js

```
var hasContentEditableSupport = function(){
return(document.getElementById("edit_profile_link").
contentEditable != null)
};

if(hasContentEditableSupport()){
$("#edit_profile_link").hide();

  var status = $("#status");
  $("span[contenteditable]").blur(function(){
    var field = $(this).attr("id");
    var value = $(this).text();

    var resourceURL = $(this).closest("ul").attr("data-url");

    $.ajax({
      url: resourceURL,
      dataType: "json",
      method: "PUT",
      data: field + "=" + value,
      success: function(data){
        status.html("The record was saved.");
      },
      error: function(data){
        status.html("The record failed to save.");
      }
    });
  });

}
```

После реализации этого решения пользователь получает возможность использовать стандартный интерфейс или более быстрый режим редактирования «на месте». Помните, что начинать следует с реализации обходного решения. Браузеры, не поддерживающие атрибут `contenteditable`, проигнорируют его (как и многие другие возможности HTML5), а пользователи не смогут работать с вашим сайтом.

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. В этой главе мы рассмотрели немало новых полей форм, но не все. Тип `search` может использоваться для полей поиска, тип `tel` — для ввода телефонных номеров, а типы `time` и `datetime` — для времени и даты со временем соответственно. Все эти разновидности полей ввода представляют пользователю конкретный интерфейс и описывают контент намного лучше, чем традиционные поля типа `text`.

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.

По мере того как встроенные средства проверки данных будут поддерживаться все большим количеством браузеров, начнут проявляться преимущества унификации; сообщения об ошибках будут понятными и последовательными, а пользователям не придется подолгу искать, где была допущена ошибка.

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.

Стилевое оформление контента и интерфейсов 4

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:

`: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]

`: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 10. 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. 8.

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. 8. 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.

css3_advanced_selectors/index.html

```
<table >
  <tr>
    <th>Item</th>
    <th>Price</th>
    <th>Quantity</th>
    <th>Total</th>
  </tr>
  <tr>
    <td>Coffee mug</td>
    <td>$10.00</td>
    <td>5</td>
    <td>$50.00</td>
  </tr>
  <tr>
    <td>Polo shirt</td>
    <td>$20.00</td>
    <td>5</td>
    <td>$100.00</td>
  </tr>
  <tr>
    <td>Red stapler</td>
    <td>$9.00</td>
    <td>4</td>
    <td>$36.00</td>
  </tr>
  <tr>
    <td colspan="3">Subtotal</td>
    <td>$186.00</td>
  </tr>
  <tr>
    <td colspan="3">Shipping</td>
    <td>$12.00</td>
    </tr>
  <tr>
    <td colspan="3">Total Due</td>
    <td>$198.00</td>
  </tr>
</table>
```

Прежде всего избавимся от уродливой стандартной сетки. Создайте новый файл с именем *stylesheets/style.css* и включите ссылку на него в страницу:

css3_advanced_selectors/index.html

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

css3_advanced_selectors/stylesheets/style.css

```
table{
  border-collapse: collapse;
  width: 600px;
}

th, td{ border: none; }
```

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

css3_advanced_selectors/stylesheets/style.css

```
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.

css3_advanced_selectors/stylesheets/style.css

```
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, `n` — счетчик, начинающийся с 0, 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)
```

Счетчик равен 0, но со смещением 2 отсчет начинается не от начала таблицы, а со второй строки.

Для выбора каждой второй строки таблицы используется множитель 2:

```
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:

css3_advanced_selectors/stylesheets/style.css

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

Мы используем два селектора, разделенных запятой, чтобы стиль применялся как к тегам `<th>`, так и к тегам `<td>`.

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.

css3_advanced_selectors/stylesheets/style.css

```
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, при помощи селектора `last-child` можно увеличить размер шрифта общей суммы в правом нижнем углу таблицы. Мы находим последний столбец последней строки и изменяем его оформление.

css3_advanced_selectors/stylesheets/style.css

```
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:

css3_advanced_selectors/stylesheets/style.css

```
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.

css3_advanced_selectors/stylesheets/style.css

```
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. 9) выглядит намного лучше, причем нам совершенно не пришлось изменять разметку.

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

Многие селекторы, использованные в решении, пока недоступны пользователям Internet Explorer; для них нужно разработать обходное решение.

Обходное решение

Селекторы, использованные в нашем решении, поддерживаются текущими версиями Internet Explorer, 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.

Ничего не делать

Простейшее решение — ничего не делать. Так как контент будет нормально читаться без дополнительного стилевого оформления, о пользователях Internet Explorer 8 можно просто забыть. Конечно, если такой подход вам кажется неправильным, существует и другое решение..

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 Selectivizr

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 Selectivizr[1], которая реализует поддержку селекторов CSS3 в Internet Explorer. Для этого необходимо лишь включить библиотеку в нашу страницу.

Библиотека Selectivizr в своей внутренней реализации может использовать jQuery, Prototype или несколько других библиотек, но jQuery поддерживает все использованные нами псевдоклассы.

Загрузите библиотеку Selectivizr и включите ссылку на нее в секцию `<head>` своего документа. Так как это решение предназначено только для Internet Explorer, поместите ссылку в условный комментарий, чтобы она загружалась только пользователями IE.

[1] http://selectivizr .com/

css3_advanced_selectors/index.html

```
<script
  src='http://ajax.googleapis.com/ajax/libs/jquery/1.9.1
                                                /jquery.min.js'>
</script>
<!--[if (gte IE 5.5)&(lte IE 8)]>
  <script src="javascripts/selectivizr-min.js"></script>
<![endif]-->
```

Обратите внимание: мы также загружаем jQuery, и в этом конкретном случае jQuery приходится загружать в секции `<head>` документа. Вызов jQuery тоже можно разместить в условном комментарии, но, скорее всего, jQuery все равно понадобится для других целей.

Включение этих сценариев в страницу решает все проблемы с оформлением таблицы в Internet Explorer. Результат показан на рис. 10.

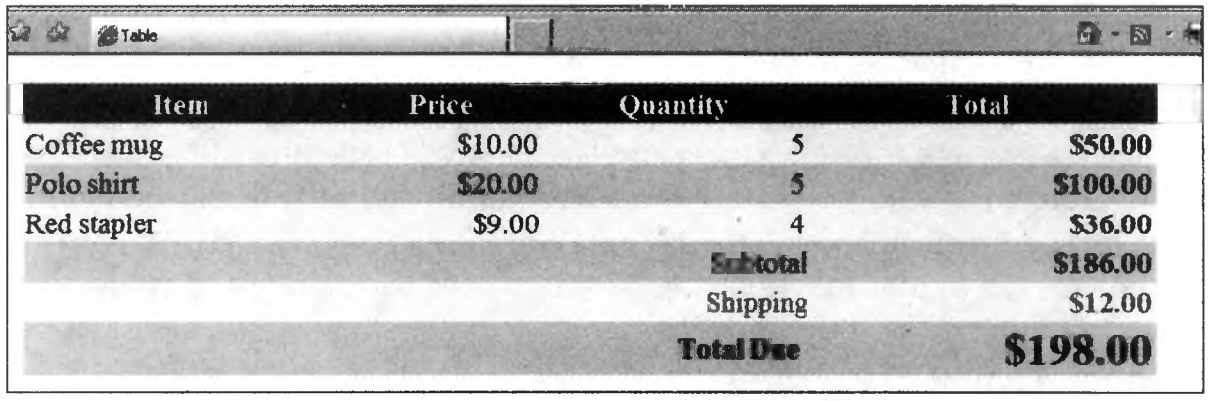


Рис. 10. Таблица нормально воспроизводится в Internet Explorer

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.

Средствами CSS также можно добавить в веб-страницу новый контент. Давайте посмотрим, как это делается.

Рецепт 11. Печать ссылок (:after)

Технология CSS обычно используется для стилевого оформления существующих документов, но она также позволяет внедрять в документ новый контент. Для этого используются псевдоэлементы `:before` и `:after` и свойство `content`. 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 перевести ссылку на другой адрес. Необходимая функциональность легко реализуется средствами CSS; такое решение будет работать в Internet Explorer 8, Firefox, Safari и Chrome.

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 текст ссылок подчеркивается, но при этом 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

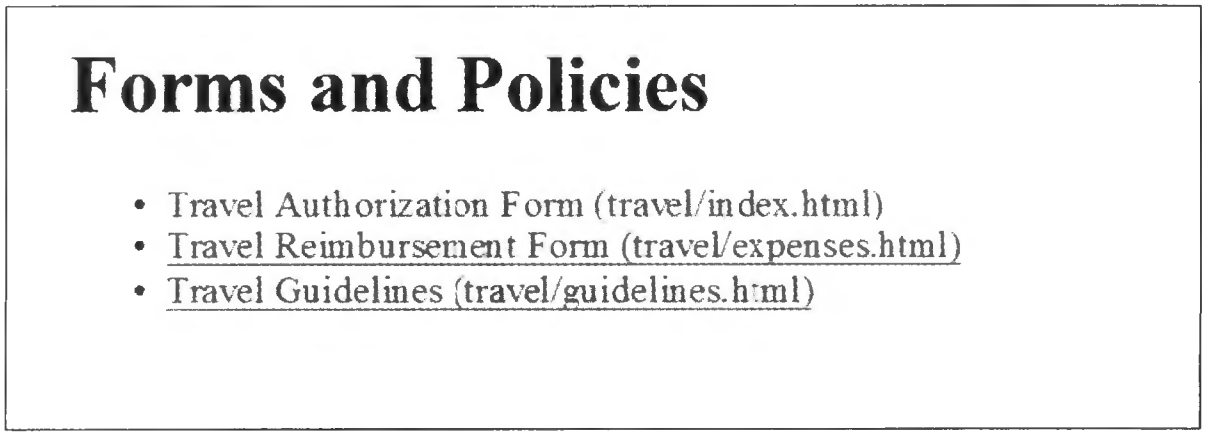
```
<link rel="stylesheet" href="print.css" type="text/css"
          media="print">
```

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

css3_print_links/stylesheets/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 из современного браузера 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.

А самое замечательное, что подобное создание контента поддерживается всеми браузерами, включая Internet Explorer 8, так что обходное решение не понадобится.

Синтаксис с двойными двоеточиями

Псевдоэлементы `:before` и `:after` появились в спецификации CSS2.1[1]. В ранних проектах они снабжались двумя двоеточиями:

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

Этот синтаксис не поддерживается многими браузерами, поэтому спецификация рекомендует разработчикам браузеров поддерживать синтаксис как с одним, так и с двумя двоеточиями.

Свойство `content` также можно использовать и другими способами. Например, оно может использоваться для создания визуальных меток в тексте. Одно из распространенных применений — размещение надписи «внешняя ссылка» рядом с URL-адресом, ведущим на другие сайты. Будьте внимательны и не переходите границу между дизайном и контентом. Свойство `content` в CSS должно использоваться только для определения дизайна страницы, а не для внедрения в нее фактического контента.

Итак, мы выяснили, как изменить внешний вид страницы при ее отправке на печать. Теперь давайте посмотрим, как изменить оформление контента в зависимости от размера экрана.

[1] http://www.w3.org/TR/CSS21/generate.html#before-after-content

Синтаксис с двойными двоеточиями

Псевдоэлементы `:before` и `:after` появились в спецификации CSS2.1[1]. В ранних проектах они снабжались двумя двоеточиями:

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

Этот синтаксис не поддерживается многими браузерами, поэтому спецификация рекомендует разработчикам браузеров поддерживать синтаксис как с одним, так и с двумя двоеточиями.

Свойство `content` также можно использовать и другими способами. Например, оно может использоваться для создания визуальных меток в тексте. Одно из распространенных применений — размещение надписи «внешняя ссылка» рядом с URL-адресом, ведущим на другие сайты. Будьте внимательны и не переходите границу между дизайном и контентом. Свойство `content` в CSS должно использоваться только для определения дизайна страницы, а не для внедрения в нее фактического контента.

Итак, мы выяснили, как изменить внешний вид страницы при ее отправке на печать. Теперь давайте посмотрим, как изменить оформление контента в зависимости от размера экрана.

[1] http://www.w3.org/TR/CSS21/generate.html#before-after-content

Рецепт 12. 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 11). Медиазапросы CSS3[1] позволяют изменять представление страницы в зависимости от размера экрана устройства 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 пользователей с различными размерами экранов. Они являются ключевой составляющей адаптивного веб-дизайна — популярной практики создания одного сайта, изменяющего свое оформление и макет в зависимости от разрешения экрана пользователя. Медиазапросы широко используются в популярных инфраструктурах — таких, как Bootstrap[2].

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.

Администрации AwesomeCo надоело слушать жалобы клиентов и работников на то, как убого смотрятся веб-страницы на смартфонах. Директор по маркетингу хочет поскорее увидеть мобильную версию шаблона блога, построенную нами для рецепта 1. Задача решается очень просто.

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

[2] http://twitter.github.com/bootstrap/

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

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

css3_mediaquery/stylesheets/style.css

```
@media only screen and (max-device-width: 480px) {
  body{ width:480px; }
    nav, section, header, footer{ margin: 0 10px 0 10px; }

    #sidebar, #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)">
```

На рис. 11 показано, как выглядит веб-страница в текущей версии. Она далека от идеала, но это хорошая отправная точка для дальнейшей работы.

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. Однако попытки уменьшить страницу, предназначенную для большого экрана, создают массу проблем. Лучше начинать проектирование с малого экрана, а потом добавить контент для экранов большего размера. Этот метод заставляет разработчика долго и напряженно думать как о контенте, так и о целевой аудитории.

Рис. 11. Страница блога на iPhone

Обходное решение

Медиазапросы поддерживаются в Firefox, Chrome, Safari, Opera, Internet Explorer 9 и выше. 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.

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

Превосходная библиотека Respond.js[1] предоставляет поддержку медиазапросов `min-` и `max-width`; это хорошее обходное решение для Internet Explorer 8, но в большинстве случаев оно не понадобится, потому что такие медиазапросы предназначены для малых экранов, а на таких устройствах Internet Explorer 8 не встречается. Тем не менее медиазапросы могут использоваться для адаптации представления для разных размеров экранов, от небольших мониторов до огромных настенных панелей.

Медиазапросы позволяют управлять отображением страницы на экранах разных размеров. Однако на больших экранах область контента может быть весьма широкой. Давайте посмотрим, как разделить ее на столбцы.

[1] https://github.com/scottjehl/Respond

Рецепт 13. 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, развернутые на всю ширину газетного листа. 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. 12.

Рис. 12. 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 не 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.

css3_columns/condensed_newsletter.html

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

```
        </div>
      </article>
      <article id="awesome_bits">
        <header>
          <h2>Quick Bits of Awesome</h2>
        </header>
        <div>
          <p>
            Lorem ipsum dolor sit amet...
          </p>
        </div>
      </article>
      <article id="birthdays">
        <header>
          <h2>Birthdays</h2>
        </header>
        <div>
          <p>
            Lorem ipsum dolor sit amet...
          </p>
        </div>
      </article>
    </section>
    <footer id="footer">
      <h6>
        Send newsworthy things to
        <a href="mailto:news@awesomeco.com">news@awesomeco.com</a>.
      </h6>
    </footer>
  </div>
</body>
```

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

css3_columns/stylesheets/style.css

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

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

Новая версия бюллетеня (рис. 13) 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.

Рис. 13. Новый двухстолбцовый бюллетень

Чтобы многостолбцовый формат работал в разных браузерах, необходимо многократно определить свойства, включая в каждое правило префикс конкретного типа браузера.

Префиксы

Пока комитет W3C разбирался с тем, какие возможности должны войти в спецификацию CSS, разработчики браузеров добавляли новую функциональность самостоятельно, помечая свои реализации префиксами. Некоторые из этих реализаций в конечном итоге вошли в стандарт, а использование префиксов превратилось в практику, которая продолжается и сейчас. Префиксы позволят разработчикам браузеров вводить новую функциональность до того, как она станет частью спецификации. Так как собственные реализации могут не соответствовать спецификации, разработчики браузеров могут реализовать спецификацию, при этом

также сохранив собственную реализацию. В большинстве случаев версия с префиксом разработчика соответствует спецификации CSS, но иногда встречаются различия. К сожалению, это означает, что вам придется неоднократно объявлять некоторые свойства для кажого типа браузера. Ниже перечислены самые распространенные префиксы разработчиков браузеров:

- Firefox использует префикс `-moz-`.
- Chrome и Safari, а также многие мобильные браузеры и последние версии Opera используют префикс `-webkit-`.
- Старые версии Opera используют префикс `-o-`.

Не увлекайтесь использованием реализаций с префиксами. По мере того как браузеры все больше реализуют стандарты, эти префиксы перестают быть необходимыми. и только захламляют CSS. Следите за тем, какие браузеры используют ваши посетители; удаляйте селекторы из стилевых таблиц, если необходимость в них отпала. Для проверки необходимости селекторов можно воспользоваться сервисом Can I Use...[1].

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.

Обходное решение

Столбцы CSS3 не работают в Internet Explorer 9 и более ранних версий. Наверное, вариант с отсутствием обходного решения тоже приемлем, потому что контент все равно читается. Но если вы стремитесь обеспечить совместимость между браузерами, попробуйте использовать модуль CSS3MultiColumn[2], реализующий функциональность многостолбцового вывода.

[1] http://caniuse.com/

[2] https://github.com/BetleyWhitehorne/CSS3MultiColumn

Просто загрузите модуль после стилевых таблиц, а он сделает все остальное. Так как обходное решение предназначено только для Internet Explorer 9 и ниже, можно использовать условный комментарий с кодом JavaScript, который заставляет Internet Explorer 8 распознавать наши элементы HTML5:

css3_columns/newsletter.html

```
<!--[if lte IE 9]>
<script>
  // Поддержка стилевого оформления элементов HTML5
  document.createElement("section");
  document.createElement("header");
  document.createElement("footer");
  document.createElement("article");
  document.createElement("aside");
</script>
<script src="javascripts/css3-multi-column.min.js"></script>
<![endif]-->
```

Обновите страницу в Internet Explorer; вы увидите двухстолбцовый бюллетень, как на следующем рисунке.

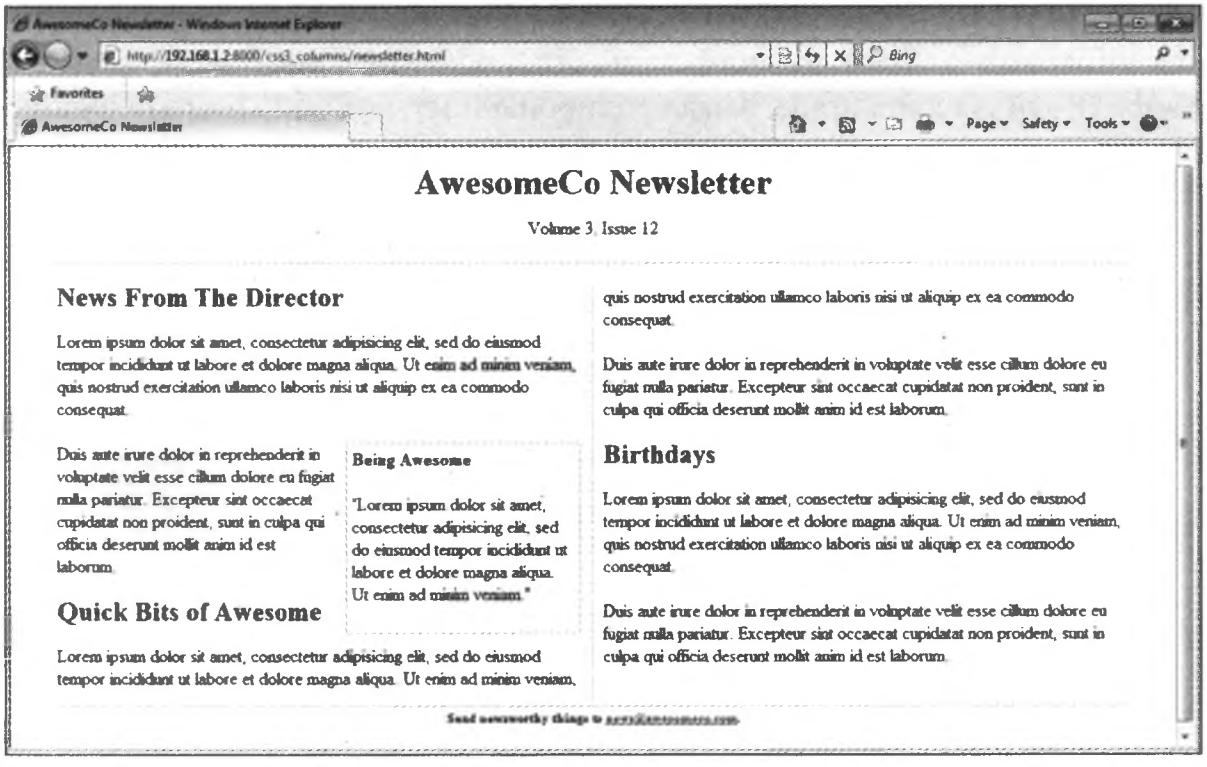


Рис. 14. Версия для Internet Explorer работает, но нуждается в доработке

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.

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.

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. Однако при этом строки таблицы не будут окрашены в чередующиеся цвета; содержимое бюллетеня будет выводиться в один столбец; пользователям придется масштабировать страницу на своих смартфонах, чтобы прочитать контент. Хорошо, что все описанные эффекты могут быть реализованы на уровне представления, без применения JavaScript или того хуже — большого объема дополнительной разметки.

В настоящее время эти селекторы поддерживаются почти всеми браузерами, за исключением Internet Explorer 8 и ниже. Когда спецификация придет к окончательному виду, специализированные префиксы вида `moz` и `webkit-` уйдут в прошлое. Когда это произойдет, код обходного решения можно будет удалить.

Улучшение доступности 5

Многие новые элементы HTML5 предназначены для более точного описания контента. Они начинают играть более важную роль, когда ваш код интерпретируется другими программами. Большинство пользователей читает веб-страницы в графических браузерах, но не стоит забывать о людях, которые взаимодействуют с ними другими способами. Мы должны позаботиться о том, чтобы сделать свой контент по возможности доступным для этой категории пользователей.

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, а затем читает его для пользователя линейно, сверху вниз.

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. А так как элементы страницы обрабатываются линейно, такие объекты, как заголовки, области навигации и виджеты, располагающиеся в начале страниц, заново читаются при каждом обновлении страницы.

Спецификация WIA-ARIA[1] (Accessibility for Rich Internet Applications) определяет способы улучшения доступности веб-сайтов и особенно веб-

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

Улучшение доступности 5

Многие новые элементы HTML5 предназначены для более точного описания контента. Они начинают играть более важную роль, когда ваш код интерпретируется другими программами. Большинство пользователей читает веб-страницы в графических браузерах, но не стоит забывать о людях, которые взаимодействуют с ними другими способами. Мы должны позаботиться о том, чтобы сделать свой контент по возможности доступным для этой категории пользователей.

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, а затем читает его для пользователя линейно, сверху вниз.

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. А так как элементы страницы обрабатываются линейно, такие объекты, как заголовки, области навигации и виджеты, располагающиеся в начале страниц, заново читаются при каждом обновлении страницы.

Спецификация WIA-ARIA[1] (Accessibility for Rich Internet Applications) определяет способы улучшения доступности веб-сайтов и особенно веб-

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

приложений. Она чрезвычайно полезна при разработке приложений с элементами JavaScript и Ajax. Некоторые компоненты спецификации WIA-ARIA были включены в HTML5[1], другие существуют отдельно и могут дополнять спецификацию HTML5. Возможности WIA-ARIA уже используются многими экранными дикторами, включая JAWS, Window-Eyes и даже встроенную функциональность Apple VoiceOver. WIA-ARIA также вводит дополнительную разметку, которая может использоваться вспомогательными устройствами для обнаружения обновляемых областей.

В этой главе вы узнаете, как HTML5 и WAI-ARIA упрощают 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:

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-hidden` [`<div aria-hidden="true">`]

Идентификация областей, которые должны игнорироваться экранным диктором. [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]

`<scope>` [`<th scope="col">Time</th>`]

Установление связи заголовка таблицы со столбцами или строками таблицы. [Все браузеры]

`<caption>` [`<caption>This is a caption</caption>`]

Создание подписи для таблицы. [Все браузеры]

`aria-describedby` [`<table aria-describedby="summary">`]

Установление связи описания с элементом. [F3.6 (Windows), S4, IE8]

[1] http://www.w3.org/TR/html5/dom.html#wai-aria

Рецепт 14. 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
application	Область страницы, содержащая веб-приложение (в отличие от веб-документа)
banner	Область баннера на странице
complementary	Информация, дополняющая основной контент, но имеющая самостоятельный смысл
contentinfo	Информация о контенте: авторское право, дата публикации и т. д.

[1] http://www.w3.org/TR/wai-aria/roles

Роль	Использование
form	Раздел страницы с формой, содержащей как элементы форм HTML, так и гиперссылки и сценарные элементы управления
main	Начало основного контента страницы
navigation	Элементы навигации на странице
search	Область поиска на странице

Некоторые из ролей можно применить к шаблону блога AwesomeCo, над которым мы работали в рецепте 1.

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 `<header>`.

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. Итак, для надежности делайте свои описания как можно более конкретными.

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="2013/10">October 2013</a></li>
      <li><a href="2013/09">September 2013</a></li>
      <li><a href="2013/08">August 2013</a></li>
      <li><a href="2013/07">July 2013</a></li>
      <li><a href="2013/06">June 2013</a></li>
      <li><a href="2013/05">May 2013</a></li>
      <li><a href="2013/04">April 2013</a></li>
      <li><a href="2013/03">March 2013</a></li>
      <li><a href="2013/02">February 2013</a></li>
      <li><a href="2013/01">January 2013</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; 2013 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.

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 поль-

зователям 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.

Роли структуры документа

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

Роль	Использование
article	Композиция, образующая независимую часть документа
definition	Определение термина или предмета
directory	Список ссылок на группу (например, оглавление). Используется для статического контента
document	Область с контентом документа (в отличие от контента приложения)
group	Совокупность объектов пользовательского интерфейса, которые не должны включаться в сводку страницы
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
toolbar	Панель инструментов в веб-приложении.

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.

В нашем блоге роль `document` логично добавить в тег `<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, поэтому роли следует протестировать с экранными дикторами в разных браузерах. Не стоит думать, будто простое назначение ролей на странице гарантирует их работоспособность во всех ситуациях.

Для тестирования рекомендуется использовать JAWS — самый распространенный экранный диктор. Хотя JAWS распространяется не бесплатно, вы можете получить демо-версию с ограниченным сроком действия[1]. Желательно протестировать JAWS с Internet Explorer и Firefox, так как в этих браузерах многие аспекты работают по-разному.

NVDA — бесплатное альтернативное решение с открытым исходным кодом — сейчас пользуется популярностью. Вероятно, его тоже следует включить в тестирование[2].

Роли помогают экранным дикторам идентифицировать важные области или элементы страницы. Они также могут передавать экранным дикторам информацию о текущем состоянии элемента. Но в современных приложениях используется динамический контент, и мы можем сообщить экранному диктору об обновлении страницы. Давайте посмотрим, как это делается.

---

[1] http://www.freedomscientific.com/downloads/jaws/jaws-downloads.asp

[2] http://www.nvda-project.org/

Рецепт 15. Создание обновляемых областей с улучшенной доступностью

JavaScript широко используется в современных веб-приложениях. Популярные инфраструктуры — такие, как Backbone и Ember — позволяют строить мощные одностраничные приложения с интерфейсами, быстро реагирующими на действия пользователя без обязательной перезагрузки страницы[1,2]. На таких страницах часто применяются визуальные эффекты, которые сообщают пользователю о неких изменениях на странице. Однако пользователь экранного диктора, естественно, этих эффектов не увидит. В прошлом такие пользователи часто отключали JavaScript в своих браузерах и пользовались обходным решением, предоставленным разработчиком, но опрос, проведенный WebAIM в 2012 году, показал, что многие пользователи экранных дикторов уже не отключают JavaScript. Это означает, что они используют тот же интерфейс, что и все остальные пользователи, поэтому мы должны каким-то образом сообщить им об изменениях в интерфейсе[3].

В спецификации WIA-ARIA представлено неплохое альтернативное решение, которое в настоящее время работает в Internet Explorer, 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. 15. 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

[1] http://backbonejs.org/

[2] http://emberjs.com/

[3] http://webaim.org/projects/screenreadersurvey4/

в события, переключающие основной контент страницы. Пользователи с включенной поддержкой JavaScript увидят, что требует заказчик, а пользователи без JavaScript увидят весь контент страницы. А самое главное, что пользователи экранных дикторов узнают об изменениях.

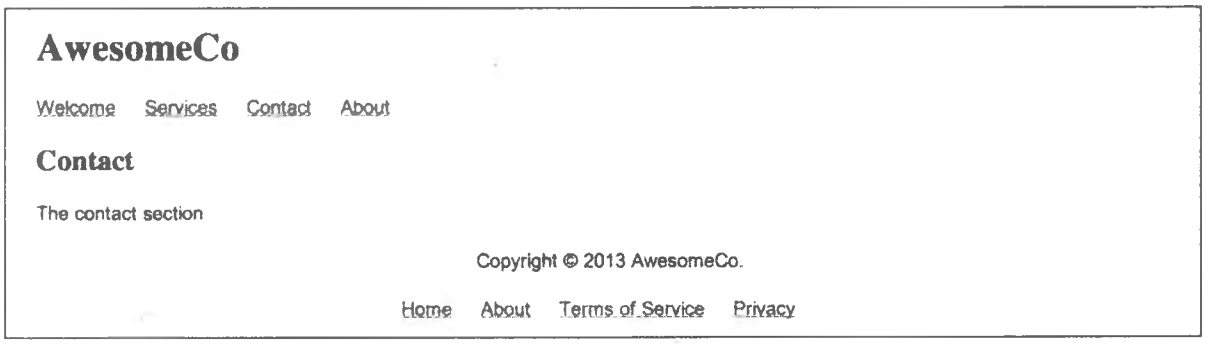


Рис. 15. Макет страницы с переключением контента средствами jQuery

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 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>Copyright © 2013 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.

Теперь добавим к странице стили CSS для создания нужного макета. Они напоминают CSS блога AwesomeCo (см. выше). Включите в файл *stylesheets/style.css* базовое стилевое оформление для тела страницы:

html5_aria/homepage/stylesheets/style.css

```
body{
  width: 960px;
  margin: 15px auto;
}

p{
  margin: 0 0 20px 0;
}

p, li{
  line-height: 20px;
  font-family: Arial, "MS Trebuchet", sans-serif;
}
```

Затем добавьте стили для создания горизонтальной области навигации в заголовке:

html5_aria/homepage/stylesheets/style.css

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

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

Напоследок определим стили для завершителя, чтобы он располагался в нижней части, а его текст был выровнен по центру.

html5_aria/homepage/stylesheets/style.css

```
footer#footer{
clear: both;
width: 100%;
display: block;
text-align: center;
}
```

Давайте посмотрим, что можно сделать с изменением внутреннего контента при щелчке на одной из ссылок.

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

<script src="javascripts/application.js">
</script>
```

Файл *application.js* содержит следующий простой сценарий:

html5_aria/homepage/javascripts/application.js

```
1  var configureTabSelection = function(){
-    $("#services, #about, #contact").hide().attr("aria-hidden",
                                                        true);
-    $("#welcome").attr("aria-hidden",false);
-
5    $("nav ul").click(function(event){
-      var target = $(event.target);
-      if(target.is("a")){
-        event.preventDefault();
-        if ( $(target.attr("href")).attr("aria-hidden") ){
10         activateTab(target.attr("href"));
-        };
-      };
-    });
-  };
15
-  var activateTab = function(selector){
-    $("[aria-hidden=false]").hide().attr("aria-hidden", true);
```

```
-     $(selector).show().attr("aria-hidden", false);
-   };
20
-   configureTabSelection();
```

В строке 2 скрываются разделы Services, About и Contact. Мы также применяем атрибут `aria-hidden` и задаем ему значение `true`. В следующей строке тот же атрибут применяется к разделу Welcome по умолчанию, но со значением `false`. Добавление атрибутов позволяет вспомогательным технологиям легко определить, какие элементы являются скрытыми и какие разделы следует включать и отключать при переключении.

В строке 5 перехватываются все щелчки в области навигации, после чего в строке 7 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 9).

Если раздел скрыт, мы вызываем метод `activateTab()`, получающий селектор CSS. Этот метод скрывает все остальное, а затем отображает выбранный раздел с использованием методов jQuery `show()` и `hide()`, а также переключает значение атрибутов `aria-hidden`.

Вот и все, что требуется сделать, — теперь 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. При внесении в пользовательский интерфейс каких-либо изменений средствами JavaScript применяйте роли ARIA к элементам, чтобы экранные дикторы обладали актуальной информацией о состоянии элемента.

На наших страницах данные часто выводятся в табличном формате. Давайте посмотрим, как обеспечить доступность таких данных.

Рецепт 16. Улучшение доступности таблиц

Таблицы HTML издавна создавали массу проблем в отношении доступности. Человеку с нормальным зрением легко охватить взглядом таблицу и получить представление о контексте. Пользователям экранных дикторов гораздо сложнее представить «общую картину». Ситуация усугублялась тем, что до того, как технология CSS начала применяться для размещения контента, разработчики использовали таблицы для определения различных областей страницы. Все это сильно затрудняло работу экранных дикторов, которые должны были перемещаться по таблицам и разбираться в том, как же их следует читать. К сожалению, даже сегодня некоторые веб-сайты применяют таблицы для формирования макетов, из-за чего в спецификацию HTML5 для таких таблиц была добавлена специальная роль ARIA:

```
<table role="presentation">
...
</table>
```

И хотя применение таблиц для управления макетом страницы — исключительно вредная привычка, потому что она заставляет смешивать представление с контентом, таблицы так часто применялись для создания макетов, что экранные дикторы стали неплохо справляться с ними. Роль `presentation` помогает немного исправить ситуацию.

Несмотря на появление новой роли, таблицы предназначены не для определения макетов, а для разметки табличных данных. Возможно, в зависимости от сложности таблицы необходимо помочь экранному диктору сообщить посетителю сайта дополнительную контекстную информацию. Для этого мы определим более четкие связи между заголовками и их строками и таблицами, а также добавим подпись и описание для таблицы.

Компания AwesomeCo проводит свою ежегодную конференцию AwesomeConf в конце декабря. На одной из страниц сайта выводится программа мероприятия в виде таблицы HTML. Вас попросили сделать так, чтобы таблица могла читаться экранными дикторами, потому что в прошлом некоторые участники в итоговом опросе жаловались на проблемы с доступностью. На рис. 16 изображена текущая версия программы конференции в таблице HTML.

Conference Schedule

Time	Room 100	Room 101	Room 152	Room 153
8:00 AM	Opening Remarks and Keynote - Ballroom			
9:00 AM	Creating Better Marketing Videos	Embracing Social Media	Pop Culture And You	Visualizing Success
10:00 AM	Build a Solid Fundraising Campaign	Print Is Dead	Mobile First? Not So Fast!	Proving What Works
11:00 AM	Making Connections	Marketing Panel	Clear Content	Improving Experiences
12:00	Lunch			

Use this grid to find the session you want to attend. Note that the keynote and lunch are in the ballroom.

Рис. 16. Страница с программой конференции

Ниже приведен фрагмент кода текущей страницы.

html5_accessible_tables/original_index.html

```
<h1>Conference Schedule</h1>

<table>
  <tr>
    <th>Time</th>
    <th>Room 100</th>
    <th>Room 101</th>
    <th>Room 152</th>
    <th>Room 153</th>
  </tr>
  <tr>
    <th>8:00 AM</th>
    <td colspan="4">Opening Remarks and Keynote - Ballroom</td>
  </tr>
  <tr>
    <th>9:00 AM</th>
    <td>Creating Better Marketing Videos</td>
    <td>Embracing Social Media</td>
    <td>Pop Culture And You</td>
    <td>Visualizing Success</td>
  </tr>
</table>
<section>
  <p>
```

```
    Use this grid to find the session you want
    to attend. Note that the keynote and lunch
    are in the ballroom.
  </p>
</section>
```

Таблица вполне стандартная, но в ней используются заголовки как столбцов, так и строк. Это обстоятельство может создать проблемы в некоторых комбинациях экранных дикторов и браузеров. Давайте обозначим связи заголовков более четко — и для нашего кода, и для экранных дикторов.

Связывание заголовков со столбцами

В простых таблицах тега `<th>` достаточно для обозначения заголовка. Браузеры и экранные дикторы используют довольно сложные алгоритмы поиска ассоциированных строк и столбцов. В более сложных таблицах можно использовать атрибут `scope`, который явно указывает, что заголовок относится к столбцу или строке. Вот как это делается:

html5_accessible_tables/accessible_index.html

```
<tr>

<th scope="col">Time</th>
<th scope="col">Room 100</th>
<th scope="col">Room 101</th>
<th scope="col">Room 152</th>
<th scope="col">Room 153</th>

</tr>

<tr>

<th scope="row">8:00 AM</th>

  <td colspan="4">Opening Remarks and Keynote - Ballroom</td>
</tr>

<tr>

<th scope="row">9:00 AM</th>

  <td>Creating Better Marketing Videos</td>
  <td>Embracing Social Media</td>
```

```
  <td>Pop Culture And You</td>
  <td>Visualizing Success</td>
</tr>
```

Для всех заголовков столбцов мы указываем атрибут `scope="col"`, а для заголовков строк — атрибут `scope="row"`. Это упрощает работу экранных дикторов, но мы также можем улучшить общую доступность таблицы, более четко описывая ее назначение.

Подписи к таблицам

Если мы представляем таблицу с информацией, желательно снабдить ее подписью, объясняющей, что делает эта таблица. Если поместить название таблицы в тег `<caption>`, экранные дикторы смогут использовать его и более четко объяснить смысл таблицы пользователю Тег `<caption>` размещается прямо под открывающим тегом `<table>`:

html5_accessible_tables/accessible_index.html

```
<caption>
<h1>Conference Schedule</h1>
</caption>

<tr>
```

ВОПРОС/ОТВЕТ

А как же атрибуты id и <headers>?

В течение многих лет правильным способом связывания заголовков таблиц со столбцами считалось назначение уникального идентификатора каждому заголовку и его включение в ячейки таблиц при помощи атрибута <headers>:

```
<table>
<tr>
<th id="name">Name</th>
<th id="email"></th>
</tr>
<tr>
<td headers="name">Ted</td>
<td headers="email">ted@puzzlesthebar.com</td>
</tr>
<tr>
<td headers="name">Barney</td>
<td headers="email">barney@puzzlesthebar.com</td>
</tr>
</table>
```

Для простых таблиц с множеством строк данных такое решение сильно увеличивает объем разметки страницы без каких-либо преимуществ перед scope. Его стоит применять только для исключительно сложных таблиц (например, содержащих вложенные заголовки). А если вы работаете с настолько сложными таблицами, подумайте, нельзя ли изменить структуру информации и сделать ее более понятной.

Иногда одной подписи недостаточно для объяснения того, что происходит в таблице. Роль `aria-describedby` позволяет связать таблицу с разделом таблицы, содержащим контент с описанием. Для нашей таблицы уже создан тег `<section>` с блоком описательного текста. Добавим в него атрибут `id`:

```
<section id="schedule_instructions">
  <p>
    Use this grid to find the session you want
    to attend. Note that the keynote and lunch
    are in the ballroom.
  </p>
</section>
```

После добавления `id` в тег `<table>` включается ссылка на раздел с описанием:

```
<table aria-describedby="schedule_instructions">
```

Включение подписей и дополнительных описаний помогает пользователям экранных дикторов лучше понять контекст таблицы, а также упрощает работу пользователей с нормальным зрением. Элемент `<caption>` поддерживается в браузерах уже несколько лет, а браузеры, не поддерживающие атрибут `aria-describedby`, просто проигнорируют его, поэтому ничто не препятствует использованию этих методов с таблицами данных.

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. Из-за простоты использования ролей их поддержка включается в популярные инфраструктуры JavaScript — такие, как Ember, jQuery Mobile и многие другие. Разработчики, использующие эти инфраструктуры, будут автоматически создавать приложения с улучшенной доступностью.

II

Графика и звук

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>`, а затем перейдем к работе с тегами `<audio>` и `<video>`. Глава завершается рассмотрением использования CSS3 для реализации теней, градиентов, трансформаций и анимаций.

6 Рисование в браузере

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.

Начнем с совместного использования JavaScript с элементом `<canvas>` на примере построения логотипа AwesomeCo из простых геометрических фигур. Затем мы воспользуемся библиотекой построения графиков, 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. Далее будет показано, как создать тот же логотип с использованием технологии SVG (Scalable Vector Graphics) — альтернативного механизма рисования в браузере. В главе рассматриваются следующие возможности:

`<canvas> [<canvas><p>Alternative content</p></canvas>]`

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

`<svg> [<svg><!-- XML content --></svg>]`

: Создание векторной графики в разметке XML. [C4, F3, S3.2, IE9, O10.1, iOS3.2, A2]

Рецепт 17. Рисование логотипа

Наше знакомство с тегом HTML5 `<canvas>` начнется с рисования простых фигур и линий. Прежде чем что-либо рисовать, необходимо включить тег `<canvas>` в страницу. Сам по себе тег `<canvas>` ничего не делает. Он всего лишь создает пустой холст, на котором можно рисовать из кода JavaScript. При создании холста указывается его ширина и высота.

html5_canvas/canvas_simple_drawing.html

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

К сожалению, ширину или высоту элемента `<canvas>` невозможно изменить средствами CSS без искажения содержимого, поэтому размеры холста должны определяться на момент объявления (или вам придется корректировать изображение на программном уровне).

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. Лучший способ — выполнение JavaScript только в том случае, если браузер поддерживает `<canvas>`. Вот как это делается:

html5_canvas/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>
}
```

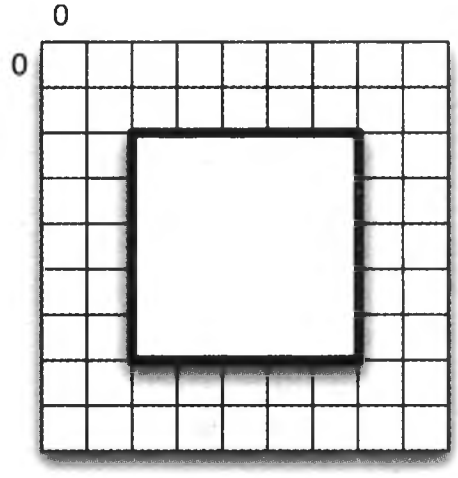
Мы находим элемент `<canvas>` по идентификатору, а затем вызываем метод `getContext()`. Если `getContext` возвращает ответ, мы получаем 2D-контекст холста для добавления объектов. Если ответа нет, то код выполняться не должен. Здесь с самого начала формируется инфраструктура для реализации обходных решений, потому что в отличие от других ситуаций, при попытке обращения к контексту в браузерах, в которых он не поддерживается, пользователи получат ошибки JavaScript.

Получив контекст холста, вы просто добавляете элементы в этот контекст, что приводит к их появлению на экране. Например, для добавления красного прямоугольника используется код следующего вида:

html5_canvas/canvas_simple_drawing.html

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

Мы устанавливаем цвет заливки, а потом создаем фигуру. 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.



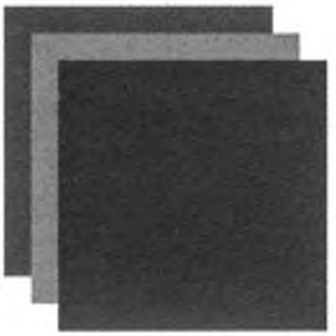
Каждая добавляемая фигура размещается на отдельном уровне; следующий фрагмент создает три прямоугольника со смещением в 10 пикселов.

html5_canvas/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.



Объединяя простые фигуры, линии, дуги и текст, можно строить довольно сложные изображения. Давайте воспроизведем на холсте логотип AwesomeCo. Логотип, приведенный на рис. 17, выглядит достаточно простым.

Рис. 17. Логотип AwesomeCo

Рисование логотипа

Логотип состоит из строки текста, ломаной линии, квадрата и треугольника. Начнем с создания нового документа HTML5. Для простоты все будет делаться в одном файле. Добавьте прямо над закрывающим тегом `<body>` блок `<script>`, в котором будет храниться код создания логотипа.

html5_canvas/logo.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AwesomeCo Logo Test</title>
  </head>
  <body>
    <script>
    </script>
  </body>
</html>
```

Затем пишется функция JavaScript для рисования логотипа. Функция проверяет возможность использования 2D-контекста.

html5_canvas/logo.html

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

Перед вызовом этого метода сначала проверяется существование элемента canvas.

html5_canvas/logo.html

```
var canvas = document.getElementById('logo');

if (canvas.getContext){
  drawLogo();
}
```

Так как код проверки ищет на странице элемент с идентификатором logo, для успешного поиска в документ необходимо добавить элемент canvas с соответствующим идентификатором.

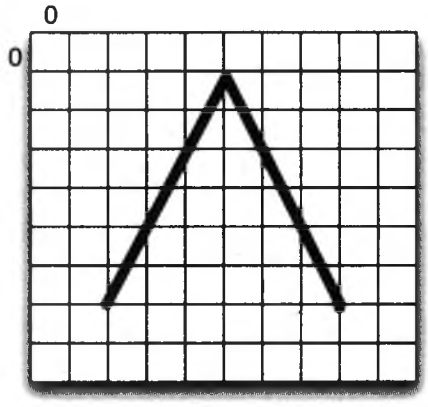
html5_canvas/logo.html

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

Переходим к рисованию логотипа.

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, как показано на следующем рисунке.



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

html5_canvas/logo.html

```
context.fillStyle = "#FF0000";
context.strokeStyle = "#FF0000";

context.lineWidth = 2;
context.beginPath();
context.moveTo(0, 40);
context.lineTo(30, 0);
context.lineTo(60, 40 );
context.lineTo(285, 40 );

context.fill();
context.closePath();
```

Прежде чем что-либо рисовать, мы назначаем цвета контура и заливки. Цвет контура определяет цвет всех линий, а цвет заливки используется для заполнения замкнутых фигур (прямоугольников, треугольников и т. д.). По сути, цвет штриха соответствует цвету периметра фигуры, а цвет заливки — цвету ее внутренней области.

После того как набор точек будет определен, вызов метода `stroke()` рисует линию, соединяющую эти точки, а вызов `closePath` прекращает рисование. Готовая линия выглядит так.

Следующий шаг — добавление в логотип надписи «AwesomeCo».

Добавление текста

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

html5_canvas/logo.html

```
context.font = "italic 40px 'Aria'";
context.fillText("AwesomeCo", 60, 36);
```

Перед нанесением текста на холст мы определяем тип текста, размер и шрифт и задаем положение базовой линии (вертикальное выравнивание). Метод `fillText()` заполняет текст цветом заливки, а вывод сдвигается на 60 пикселов вправо и 36 пикселов вниз, чтобы текст располагался в правой части только что нарисованной ломаной — прямо над нарисованной линией, рядом с большим треугольником.

Остается нарисовать рамку и комбинацию из треугольников внутри большого треугольника.

Перемещение начала координат

Вместо сложной фигуры из двух треугольников мы нарисуем маленький квадрат, а затем наложим на него белый треугольник. 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. Перемещение начала координат упрощает рисование новых фигур в нужном месте и избавляет от необходимости прибавления смещений к координатам всех точек фигуры.

Рисуем меньший внутренний квадрат со смещением начала координат.

html5_canvas/logo.html

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

Квадрат размещается внутри треугольного изгиба ломаной.

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.

html5_canvas/logo.html

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

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 градиента

Цвета штриха и заливки задаются перед началом рисования:

html5canvasgraph/logo.html

```
context.fillStyle = "#FF0000";
context.strokeStyle = "#FF0000";
```

Впрочем, это слишком банально. Интереснее создать градиенты и назначить их линиям и заливкам.

html5_canvas/logo_gradient.html

```
var gradient = context.createLinearGradient(0, 0, 0, 40);
gradient.addColorStop(0, "#AA0000"); // темно-красный
gradient.addColorStop(1, "#FF0000"); // 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 (пожалуйста, *не делайте* этого в своих страницах!).

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

Компания Google выпустила библиотеку ExplorerCanvas[1], благодаря которой большая часть Canvas API становится доступной для пользователей Internet Explorer. На момент написания книги самая стабильная версия 3.0 не поддерживала добавление текста, а библиотека не обновлялась с 2009 года. Соответственно мы воспользуемся версией из репозитория Subversion, которая работает намного лучше, но все равно обладает некоторыми ограничениями (о которых можно прочитать в исходном коде библиотеки[2]). Чтобы использовать библиотеку, мы включаем ее в секцию `<head>` страницы.

html5_canvas/logo_gradient.html

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

Затем перед проверкой холста добавляется следующий фрагмент:

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

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

html5_canvas/logo_gradient.html

```
var canvas = document.getElementById("logo");

var G_vmlCanvasManager; // Чтобы избежать ошибок в других
                        браузерах
if (G_vmlCanvasManager != undefined) { // IE 8
  G_vmlCanvasManager.initElement(canvas);
}

if (canvas.getContext){
  drawLogo();
}
```

Этот фрагмент заставляет библиотеку ExplorerCanvas присоединить свою функциональность к определенному нами элементу `canvas`. Иногда библиотека ExplorerCanvas не успевает завершить свои манипуляции с DOM к тому моменту, когда мы уже готовы использовать ее. Если использовать jQuery и поместить функцию `drawLogo()` в обработчик jQuery `document.ready()`, этот фрагмент не понадобится.

После таких изменений все отлично работает в Internet Explorer 8.

Для такого простого логотипа также можно было просто разместить в теге `<canvas>` графику логотипа в формате PNG. Браузеры, в которых `<canvas>` не поддерживается, просто выведут изображение.

Мы рассмотрели возможности рисования простых фигур. Теперь посмотрим, как сделать что-нибудь посложнее.

Рецепт 18. Построение диаграмм средствами RGraph

Тег <canvas> отлично подходит для рисования простой графики, но сама возможность размещения объектов из JavaScript означает, что его также можно использовать для визуализации данных. В этом рецепте мы используем <canvas> для создания простой диаграммы.

Существует много разных средств для построения диаграмм и графиков на веб-страницах. В прошлом разработчики часто использовали 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. С библиотекой RGraph[1] задача 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.

html5_canvas/rgraph_bar_example.html

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

<script src="javascripts/RGraph.common.js" ></script>
<script src="javascripts/RGraph.bar.js" ></script>

<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"]);
```

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

```
  bar.Draw();
</script>
```

От вас потребуется лишь создать пару массивов JavaScript, а библиотека построит диаграмму на холсте сама. Диаграмма, построенная по этим данным, изображена на рис. 18.

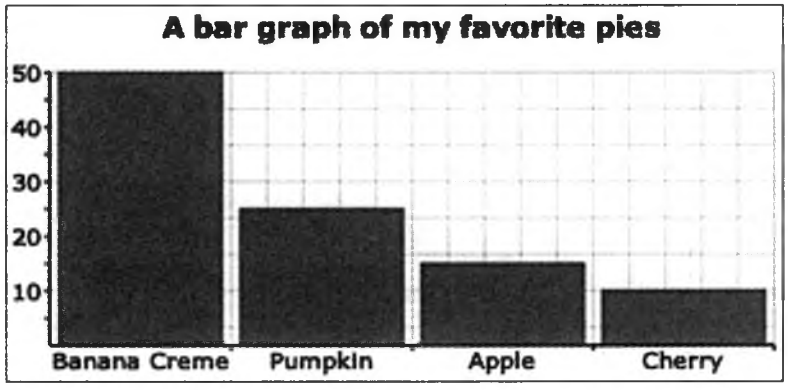


Рис. 18. Построение диаграммы на стороне клиента с использованием элемента canvas

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. 18.

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.

html5_canvas/canvas_graph.html

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

```
        Safari - 10%
      </p>
    </li>
    <li>
      <p data-name="Internet Explorer" data-percent="55">
        Internet Explorer - 30%
      </p>
    </li>
    <li>
      <p data-name="Firefox" data-percent="14">
        Firefox - 15%
      </p>
    </li>
    <li>
      <p data-name="Google Chrome" data-percent="16">
        Google Chrome - 45%
      </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. 19, 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.

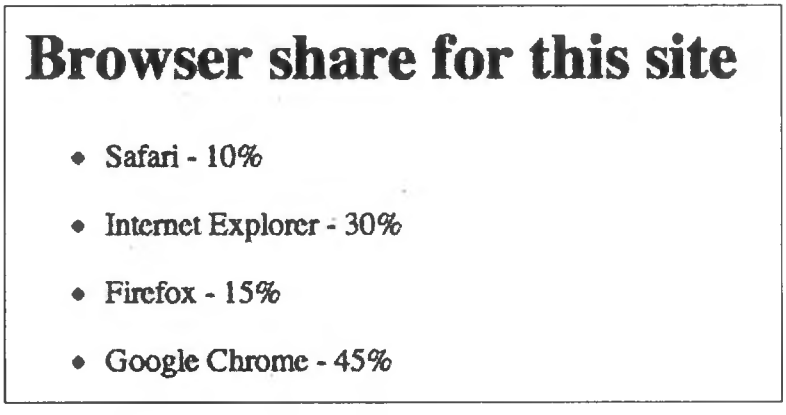


Рис. 19. Наша диаграмма в виде HTML

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. Код построения диаграммы будет вынесен в файл с именем *javascripts/graph.js*.

Необходимые библиотеки загружаются непосредственно перед закрывающим тегом `<body>`:

html5_canvas/canvas_graph.html

```
<script
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1
                                                /jquery.min.js">
</script>
<script src="javascripts/RGraph.common.js" ></script>
<script src="javascripts/RGraph.bar.js" ></script>
<script src="javascripts/graph.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. Добавьте в файл *javascripts/graph.js* следующий код:

html5_canvas/javascripts/graph.js

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

В строке 2 мы читаем текст заголовка диаграммы. Далее, в строке 3 выбираются все элементы с атрибутом `data-name`. Для преобразования значений этих элементов в массив используется функция jQuery `map`.

Аналогичная логика используется в строке 6 для заполнения массива процентов.

В строке 7 значение атрибута преобразуется в целое число. Также можно было воспользоваться методом jQuery `data()`, который читает атрибуты данных HTML5 и автоматически преобразует значение к соответствующему типу.

После того как все данные будут собраны, RGraph легко строит диаграмму (рис. 20).

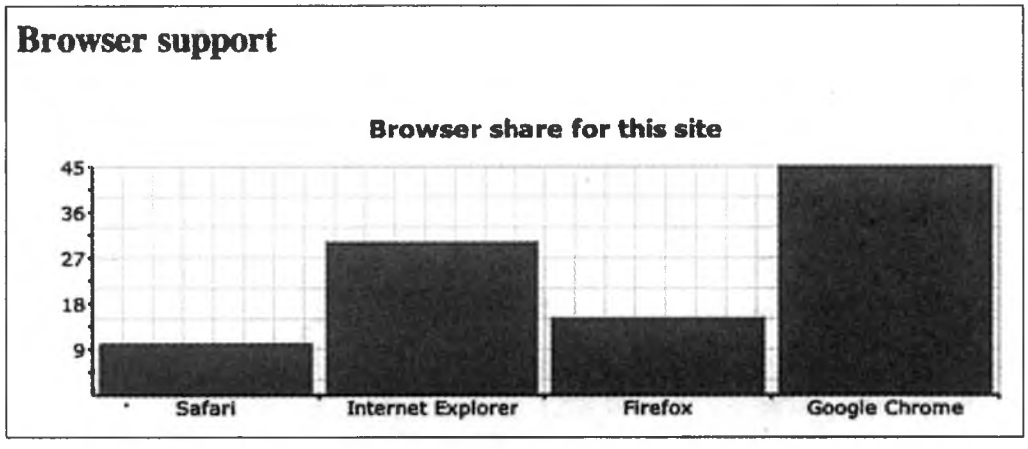


Рис. 20. Диаграмма, выведенная на элементе canvas

Отображение альтернативного контента

В разделе «Описание данных в HTML» 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.

Соответственно мы оставляем данные за пределами элемента `canvas`, а затем скрываем их средствами jQuery после проверки поддержки `canvas`. Для проверки вместо Modernizr будет использоваться стандартный код JavaScript, потому что это делается очень просто.

html5_canvas/javascripts/graph.js

```
var canvas = document.getElementById('browsers');
if (canvas.getContext){
  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 наша диаграмма не будет отображаться в Internet Explorer 8, потому что этот браузер не поддерживает `canvas`.

Библиотеки ExplorerCanvas (см. рецепт 17) и RGraph хорошо сочетаются друг с другом. Достаточно включить библиотеку *excanvas.js* в секцию `<head>`, чтобы диаграммы автоматически заработали в Internet Explorer 8. Но если вы работаете в Internet Explorer 7 и ранее, придется использовать альтернативное решение.

Поскольку библиотека ExplorerCanvas должна загружаться в секции `<head>`, в отдельных случаях может возникать ситуация гонки при загрузке библиотек в неверном порядке. Библиотека ExplorerCanvas должна внести изменения в модель DOM, а эти изменения не всегда вносятся так быстро, как нам хотелось бы. Проблему можно решить двумя способами. Во-первых можно использовать конфигурацию из раздела «Обходное решение» на с. 148, чтобы заставить ExplorerCanvas увидеть наш элемент `canvas`. Во-вторых, можно воспользоваться методом jQuery `$(document).ready()` для вызова функции `canvasGraph()`. Это гарантирует, что документ действительно готов для манипуляций со стороны наших сценариев. Библиотека jQuery все равно используется в приложении, там что остается лишь слегка изменить код:

html5_canvas/javascripts/graph.js

```
$(document).ready(function(){
  var canvas = document.getElementById('browsers');
  if (canvas.getContext){
    canvasGraph();
  }
});
```

Теперь все заработало и в Internet Explorer 8!

Использование `canvas` имеет дополнительное преимущество — оно заставляет нас думать об обходном решении с самого начала вместо того, чтобы попытаться что-нибудь втиснуть в готовое решение. Это один из самых доступных и универсальных способов графического представления данных; вы можете легко создать как визуальное представление, так и текстовую альтернативу. В этом случае все пользователи смогут понять важные данные, которые вы им передаете.

А теперь рассмотрим совершенно иной механизм рисования в браузере.

Рецепт 19. Создание векторной графики SVG

Возможности вывода графики не ограничиваются рисованием на холсте. Документы HTML5 поддерживают графику SVG (Scalable Vector Graphics). Вместо того чтобы чертить линии и рисовать фигуры из JavaScript, можно определять линии, кривые, круги, прямоугольники и многоугольники в XML. Графика SVG является полноценной векторной графикой — вместо построения изображений из пикселов (как в растровой графике) для определения контуров используются математические описания. Это означает, что векторную графику можно легко масштабировать без размывки или потери качества (в отличие от растровой графики `<canvas>`).

Чтобы поближе познакомиться с SVG, мы используем XML-синтаксис SVG для создания логотипа AwesomeCo из предыдущего рецепта.

Начнем с создания простой заготовки HTML с элементом `<svg>`:

html5_svg/index.html

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset="utf-8">
    <title>AwesomeCo Logo Test</title>
  </head>
  <body>
    <script type="image/svg+xml">
      <svg id="awesomeco_logo" width="900" height="80">
      </svg>
    </script>
  </body>
</html>
```

Тег `<svg>` определяется в теге `<script>` с типом контента `image/svg+xml`. Это гарантирует, что контент SVG будет пропускаться браузерами при разборе элементов HTML-страницы. В теге `<svg>` также указывается ширина и высота элемента SVG. Разметка XML, определяющая изображение, которое мы создаем, заключается между открывающим и закрывающим тегами `<svg>`.

Рисование линий

Рисование логотипа начнется с главной линии, на которой располагается текст. В SVG для этой цели существует тег `<<polyline>`, предназначенный для создания линий с изломами. Главная линия вместе с изломом рисуется следующей разметкой:

html5_svg/index.html

```
<polyline id="line"
          points="0,40 30,0 60,40 285,40"
          style="fill:none;stroke:rgb(255,0,0);stroke-width:2">
</polyline>
```

Как и в случае с `<canvas>`, в начале рисования точка с координатами $x = 0$ и $y = 0$ находится в левом верхнем углу. Чтобы определить линию с использованием тега `<polyline>`, мы задаем набор точек. Первая точка смещена на 0 пикселов вправо и на 40 пикселов вниз. Затем линия перемещается в точку на 30 пикселов вправо и 0 пикселов вниз от начала координат; так создается первый отрезок. Следующая точка смещена на 60 пикселов вправо и 40 пикселов вниз, а последняя точка — на 285 пикселов вправо и 40 пикселов вниз.

Атрибут `style` определяет толщину и цвет линии. После того как линия будет определена, результат выглядит примерно так:

Далее можно переходить к добавлению текста.

Добавление текста

Для определения выводимого текста используется тег `<text>`. Атрибуты этого тега определяют семейство и размер шрифта, его начертание и насыщенность.

В тег `<text>` включается элемент `<tspan>`, непосредственно создающий блок текста. Его позиция задается атрибутами `x` и `y`, а выводимый текст заключается между открывающим и закрывающим тегами `<tspan>`. Чтобы текст логотипа выводился в нужном нам виде, его следует закодировать следующим образом:

html5_svg/index.html

```
<text id="AwesomeCo"
      fill="rgb(255,0,0)"
      font-family="Arial" font-size="40"
      font-style="italic" font-weight="normal">
      <tspan x="60" y="36" fill="rgb(255,0,0)">AwesomeCo</tspan>
</text>
```

Текст правильно размещается в логотипе:

Переходим к созданию меньшего прямоугольника и треугольника.

Добавление фигур

В SVG поддерживаются определения кругов, эллипсов и даже неправильных многоугольников. Квадрат определяется тегом `<rect>`. При создании квадрата определяются начальные координаты, ширина и высота.

html5_svg/index.html

```
<rect id="square"
  x="20" y="20" height="20" width="20"
style="fill:rgb(255,0,0)"></rect>
```

Цвета штриха и заливки определяются точно так же, как это делалось для `<polyline>` — с указанием атрибута `style`. Наш логотип постепенно обретает форму:

Остается добавить маленький белый треугольник.

Рисование объектов произвольной формы

Для определения треугольника можно использовать тег `<polygon>`, но вместо этого мы используем тег `<path>` аналогично тому, как это делалось при рисовании исходного треугольника в версии `<canvas>` логотипа. Решение получается действительно очень похожим, разве что все перемещения и точки описываются одним массивом координат, разделенных пробелами (вместо серии шагов).

Прорисовка линии определяется атрибутом `d`. Мы начнем с определения начальной точки линии, а затем последовательно нарисуем каждую точку. Так как рисуется замкнутая фигура, последняя точка пути должна совпадать с исходной.

html5_svg/index.html

```
<path id="Triangle"
  d="M20,40 L30,20 L40,40 Z M20,40"
fill="rgb(255,255,255)"></path>
```

Операция *M* осуществляет перемещение относительно начала координат. Операция L задает точку на линии, а операция Z замыкает линию (начальная точка соединяется с конечной для заполнения внутренней части фигуры). Представьте, что вы развлекаетесь с головоломкой с соединением точек: сначала перо перемещается к первой точке, затем проводится отрезок к следующей точке и т. д.

После завершения рисования логотип выглядит так:

Графику SVG не обязательно встраивать в документ так, как это сделано у нас. Ее можно определить во внешнем файле и даже загрузить как фоновое изображение в CSS (при условии, что ваш браузер поддерживает такую возможность).

Рисование в SVG напоминает рисование на элементе `<canvas>`, но у разработчика имеется дополнительная возможность определить изображение в отдельном файле и связать его с документом. Таким образом, эта технология отличается большей гибкостью. С другой стороны, она не так хорошо подходит для программирования. Графика SVG более уместна в логотипах и нединамическом контенте, тогда как графика `<canvas>` может использоваться в играх. Теперь, когда мы рассмотрели оба варианта, вы можете заняться самостоятельными исследованиями и найти тот вариант, который лучше подойдет для ваших целей.

Обходное решение

Браузеры, не обладающие встроенной поддержкой SVG, могут использовать разнообразные обходные решения. Проще всего реализуется вариант с библиотекой SVG Web[1], обеспечивающей частичную поддержку SVG 1.1. Для вывода векторной графики SVG Web использует Flash.

Загрузите SVG Web и поместите файлы *svg.js* и *svg.swf* в папку *javascripts*. Включите ссылку на *svg.js* в веб-страницу в теге `<script>`, находящемся в секции `<head>` документа:

[1] https://code.google.com/p/svgweb/

html5_svg/index.html

```
<script src="javascripts/svg.js" data-path="javascripts">
</script>
```

Вот и все, обходное решение работает.

Перспективы

Теперь вы примерно представляете, как работает элемент `<canvas>`, и можете найти новые возможности его использования. Например, на базе `<canvas>` можно создать игру с использованием таких библиотек, как Impact[1] или Crafty[2], интерфейс для воспроизведения мультимедийных файлов или графическую галерею с расширенными возможностями. Все, что нужно для рисования — JavaScript и немного воображения. А по мере совершенствования поддержки скорость и функциональность графических средств будут только улучшаться.

Однако возможности `<canvas>` не ограничиваются 2D-графикой. Спецификация `<canvas>` предусматривает поддержку 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. Ознакомьтесь с превосходной библиотекой Three.js и посмотрите, какие замечательные творения в ней можно создать[3].

Что касается SVG, такие библиотеки, как Raphaël[4], позволяют легко создавать впечатляющие визуализации и графику, работающие во всех браузерах. А самое замечательное, что многие программы (например, Adobe Illustrator) позволяют экспортировать векторную графику в файлы SVG для последующего включения в веб-проекты. По мере совершенствования поддержки SVG станет еще проще создавать изображения, масштабируемые по размерам устройства. В мире, в котором многие пользователи часто переключаются между малыми экранами мобильных устройств и большими экранами настольных компьютеров, возможность создания графики, масштабируемой без потери качества, будет очень полезной.

---

[1] http://impactjs.com/

[2] http://craftyjs.com/

[3] http://threejs.org/

[4] http://raphaeljs.com

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 (например, плагин Flash). 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 элементы:

`<audio>` [`<audio src="drums.mp3"></audio>`]
: Воспроизведение аудио встроенными средствами браузера. [C4, F3.6, S3.2, IE9, O10.1, IOS3, A2]

`<video>` [`<video src="tutorial.m4v"></video>`]
: Воспроизведение видео встроенными средствами браузера. [C4, F3.6, S3.2, IE9, O10.5, IOS3, A2]

`<source>` [`<source src="video/h264/01_blur .mp4" type='video/mp4'>`]
: Определение исходного аудио- или видеофайла; используется для разных форматов. [C4, F3.6, S3.2, IE9, O10.5, IOS3, A2]

`<track>`
: Определение субтитров, подписей или разбивки на главы для видеоматериалов. [C18, S6.1, IE10]

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.

7.1. Немного истории

Попытки использования аудио и видео в веб-страницах начались довольно давно. Сначала разработчики внедряли MIDI- или MP3-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 вошел в стандарт, 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. Все это было крайне неудобно для разработчиков и производителей контента, а для рядового пользователя и вовсе превращалось в сущий кошмар.

Компания Macromedia (ныне Adobe) довольно быстро осознала, что ее Flash Player может стать идеальным кросс-платформенным средством воспроизведения аудио- и видеоконтента. Технология Flash поддерживалась 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 (а позднее и на iPad). Многие поставщики контента, включая Youtube, отреагировали на эту новость созданием видеопотоков, нормально вос-

производимых в браузере Safari для iOS. 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. Браузеры должны относиться к аудио и видео как к полноправным видам веб-контента (наряду со статической графикой). Но прежде чем рассматривать возможности встраивания аудио- и видеоматериалов в страницу, необходимо немного поговорить о форматах.

7.2. Контейнеры и кодеки

При обсуждении работы с видео в Web часто используются термины «контейнер» и «кодек». Видеоролик, поступающий с цифровой камеры, можно представить себе в виде файла в формате AVI или MPEG, но это чрезвычайно упрощенный взгляд. *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. Все эти форматы немного отличаются друг от друга, и к сожалению, разные браузеры поддерживают разные форматы.

Видеокодеки и их поддержка браузерами

H.264
[C3, F21 (Windows 7+), S4, IE9, iOS]

Theora
[F3.5, C4, O10]

VP8
[C5, F4, S6 и IE9 (если кодек установлен), 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. По этой причине кодек Theora распространения не получил и был заменен форматом VP8.

VP8

VP8 фирмы Google — полностью открытый бесплатный кодек, по качеству сходный с H.264. Он поддерживается в Mozilla, 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. Кроме того, VP8 может нарушать патентное право, связанное с кодеком H.264[1].

Аудиокодеки

Ситуация с конкуренцией стандартов видеоконтента дополнительно усложняется наличием конкурирующих аудиостандартов.

Аудиокодеки и их поддержка браузерами

AAC
[S4, C3, IOS]

MP3.
[C3, S4, IE9, IOS]

[1] http://www.fosspatents.com/2013/03/nokia-comments-on-vp8-patent.htm

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.

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.

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.

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, а также Internet Explorer 9 и выше. 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 — в формате H.264 для Safari, iOS и Internet Explorer 9 и выше и в формате VP8 для 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 20. 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 в теге `<audio>` определяются внутренние теги `<source>`: для MP3- и OGG-версии семпла. Атрибут `type` описывает тип аудио; если задать его, браузеру не придется обращаться к серверу за информацией о формате видео; он может просмотреть поддерживаемые типы и очень быстро выполнить проверку. Если он не может воспроизвести первый найденный файл, он опробует следующий и так далее, пока не будут обработаны все теги `<source>`.

Наконец, ссылка для загрузки файла MP3 включается 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. 21). 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?

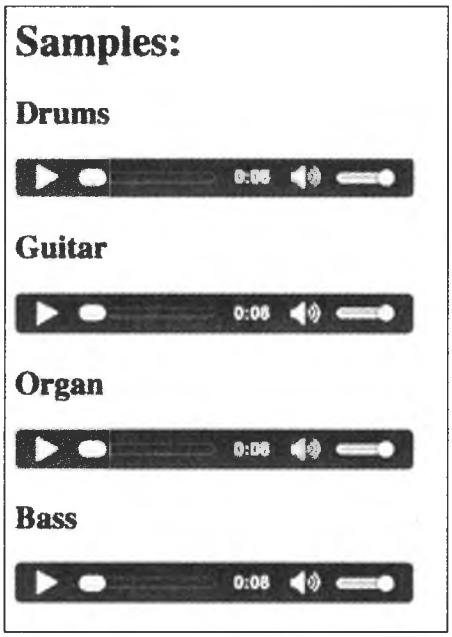


Рис. 21. Страница с семплами в Chrome

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>
```

Обходные решения для работы с аудио реализуются относительно просто. Вероятно, некоторым из ваших пользователей даже понравится возможность просто загрузить нужный файл, и вы вообще не захотите скрывать ссылки.

Если вы захотите сделать что-то менее тривиальное, вам понадобится проверить поддержку аудио. Простейший способ проверки — создание нового элемента `audio` в JavaScript и проверка его реакции на метод `canPlayType()`:

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

Чтобы проверить поддержку определенного типа аудио, используйте метод `canPlayType()` с элементом `audio`:

```
var audio = document.createElement('audio');
if(audio.canPlayType('audio/ogg')){
  // Воспроизводит файлы ogg
}
```

Метод `canPlayType()` не возвращает логические значения. Вместо `true` или `false` вы получаете одну из следующих строк:

- `"probably"` — скорее всего, будет работать.
- `"maybe"` — не исключено, что будет работать.
- `""`, пустая строка, которая интерпретируется как ложное значение. Таким образом, хотя значение и не является классическим булевским `false`, JavaScript не будет воспринимать его как `true`.

В Modernizr реализована чуть более совершенная поддержка доступности аудио. Библиотека предоставляет удобные вспомогательные методы, использующие `canPlayType()` в своей внутренней реализации:

```
if(Modernizr.audio.ogg){
  // Воспроизводит файлы ogg
}

if(Modernizr.audio.mp3){
  // Воспроизводит файлы MP3
}
```

Однако для проверки аудиоформатов, воспроизводимых в браузере, Modernizr продолжает использовать возвращаемые значения спецификации HTML5.

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. 183).

Теперь, когда вы знаете, как использовать встроенную поддержку аудио, давайте посмотрим, как сделать то же самое с видео.

Рецепт 21. 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 (за дополнительной информацией о самостоятельном кодировании видеофайлов обращайтесь к приложению B).

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

html5_video/index.html

```
<article>
  <header>
    <h2>Saturate with Blur</h2>
  </header>
  <video id="video_blur" preload="auto" controls
         width="640" height="480">
    <source src="video/h264/01_blur.mp4" type='video/mp4'>
    <source src="video/theora/01_blur.ogv" type='video/ogg'>
    <source src="video/webm/01_blur.webm" type='video/webm'>
    <p>Your browser does not support the video tag.</p>
  </video>
</article>
```

Разметка определяет тег `<video>` с элементами управления воспроизведением. Атрибут `preload` приказывает браузеру попытаться загрузить ролик в фоновом режиме, а *отсутствие* атрибута `autoplay` неявно указывает на то, что ролик не должен воспроизводиться автоматически.

В теге `<video>` мы определяем каждый видеоисточник `<source>` и указываем его тип. Браузер воспроизводит файл того типа, который он поддерживает, но мы должны правильно сообщить браузеру информацию о формате. Если указать только формат MP4, то браузеры, поддерживающие тег `<video>`, но не поддерживающие MP4, покажут пользователю пустой видеопроигрыватель.

Чтобы веб-сервер знал, как обрабатывать наши видеофайлы, необходимо добавить в него соответствующие типы MIME. Конкретные действия

зависят от платформы, но при использовании Apache следует создать в той папке, где находится веб-страница, новый файл *.htaccess* с определениями MIME-типов для видео.

html5_video/.htaccess

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

Если вы поставляете видео с Amazon S3, задайте заголовок `content-type` для каждого видеоролика. А если вы используете Microsoft Internet Information Server, отредактируйте типы MIME вашего сайта в его интерфейсе.

После отправки файлов на веб-сервер и настройки типов MIME видеоролик, оформленный таким образом, будет воспроизводиться в широком спектре браузеров. Примерный вид проигрывателя показан на рис. 22.

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

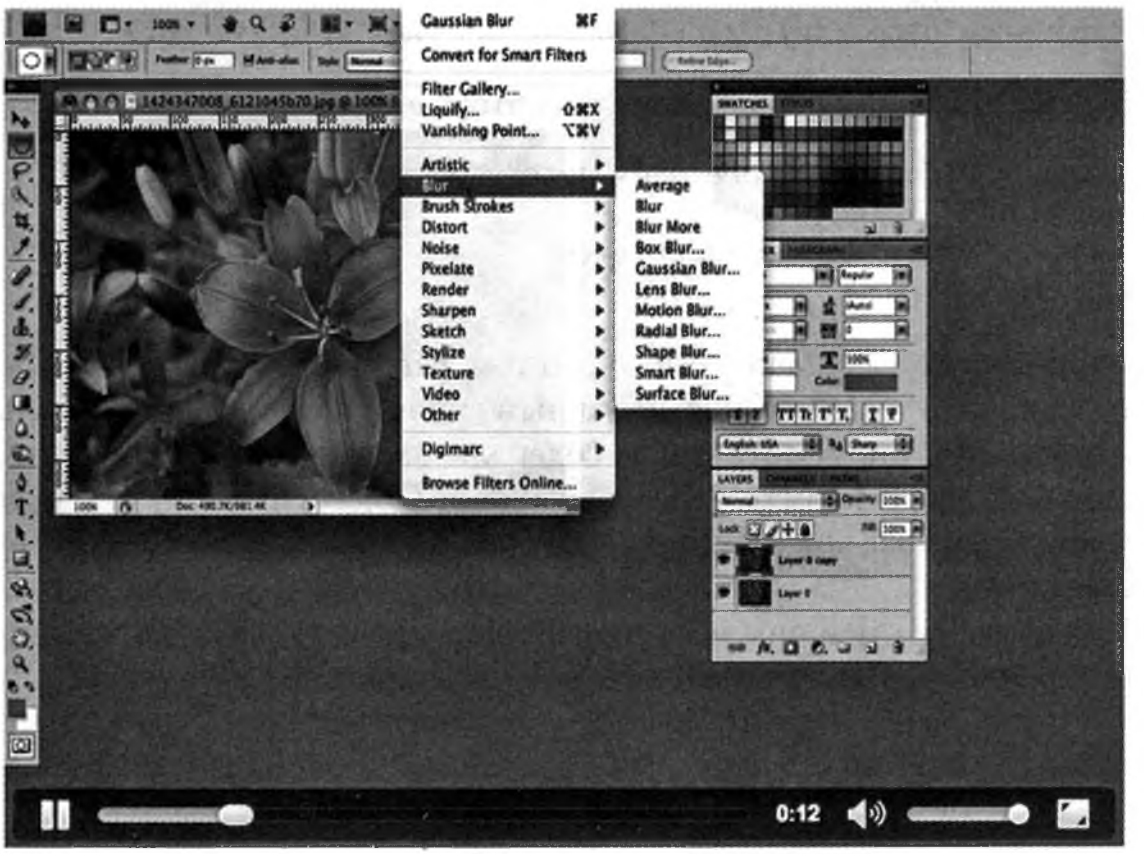


Рис. 22. Воспроизведение видеоролика во встроенном видеопроигрывателе Chrome

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 достаточно подробно, но существует более простой способ воспроизведения видео на разных платформах без написания громоздкой разметки.

Библиотека Video.js[2] обеспечивает невероятно простую поддержку видео на всех платформах. Информация, содержащаяся в теге `<video>`, используется для построения подходящего обходного решения на базе Flash для браузеров без поддержки видео HTML. Ниже приводится краткое описание использования этой библиотеки.

Сначала следует добавить библиотеку Video.js и таблицу стилей в существующую страницу. Video.js предоставляет в ваше распоряжение сеть доставки контента, так что загружать библиотеки самостоятельно не нужно. Достаточно добавить следующий код в секцию `<head>` страницы:

html5_video/videojs_index.html

```
<link href="http://vjs.zencdn.net/4.0/video-js.css"
rel="stylesheet">
<script src="http://vjs.zencdn.net/4.0/video.js"></script>
```

Затем в тег `<video>` добавляются некоторые атрибуты:

```
<video id="video_blur" preload="auto" controls
  width="640" height="480"
  class="video-js vjs-default-skin"
  data-setup="{}">
```

Атрибут `class` сообщает Video.js о том, что библиотека должна использовать этот видеоролик с указанным скином проигрывателя. Пользовательский атрибут `data-setup` содержит параметры конфигурации, представленные в формате JSON (JavaScript Object Notation). В нашем случае специальная конфигурация не нужна, поэтому мы просто передаем пустой объект.

При загрузке страницы в Internet Explorer видео успешно воспроизводится, причем нам не нужно выполнять кодирование в *другой* формат или загружать собственный проигрыватель на базе Flash. Пользователи Internet Explorer видят проигрыватель, показанный на рис. 23. Важно

[1] http://camendesign.com/code/video_for_everybody

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

учитывать, что настройки безопасности Flash могут запретить просмотр видео, если только HTML и видео не поставляются с веб-сервера. URL-адрес с префиксом `file:` может не сработать.

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

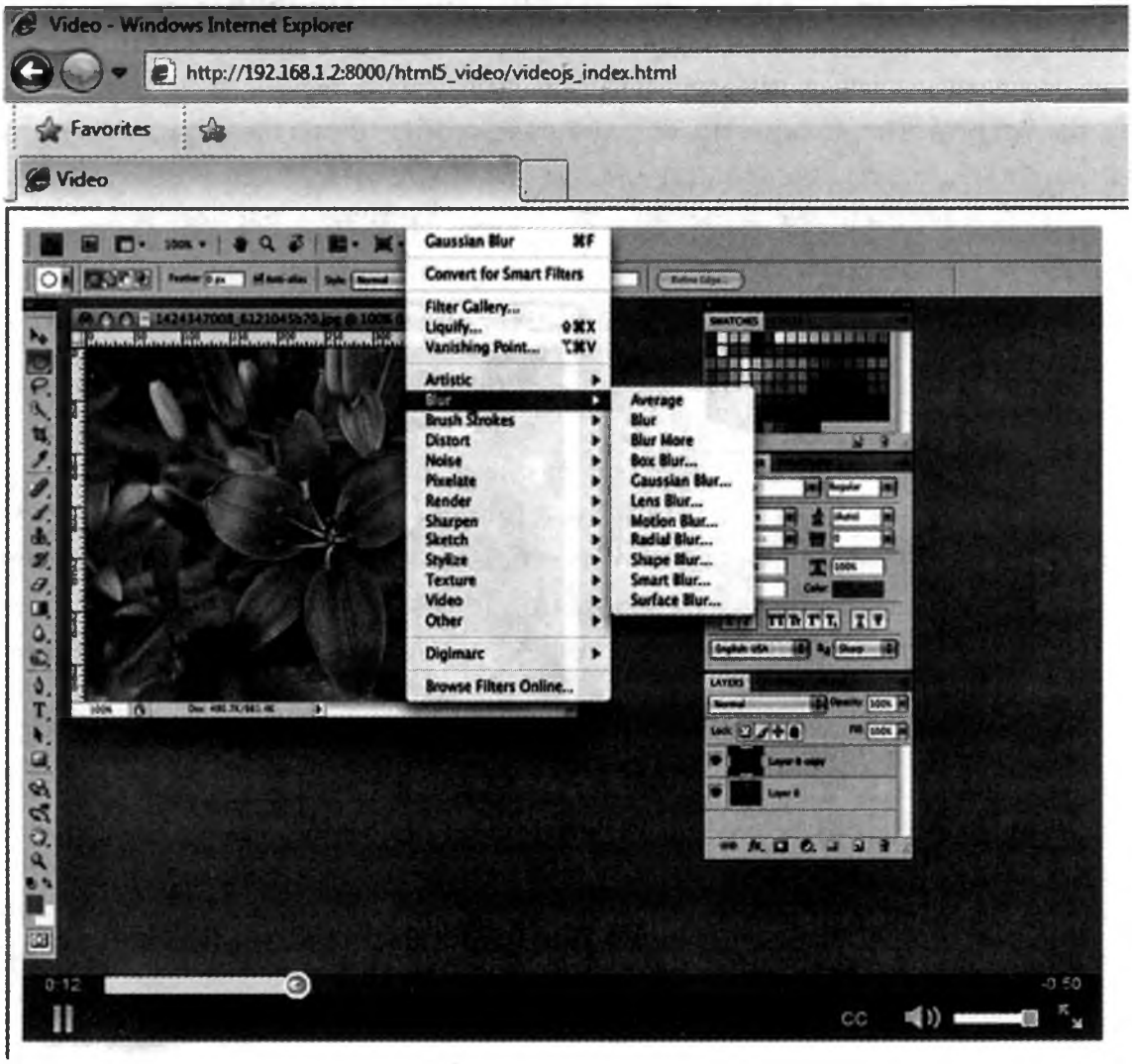


Рис. 23. Воспроизведение видео в Internet Explorer с использованием Video.js

html5_video/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>
```

Для проверки поддержки видео можно было использовать Modernizr по аналогии с тем, как было сделано в рецепте 20. Но во многих случаях будет разумнее дать возможность людям загрузить видеоролики для автономного использования, чтобы они могли просмотреть их позднее на планшете или другом устройстве. Сокрытие ссылок сэкономит место на экране, но не помешает технически компетентным людям загрузить видео напрямую, так что не рассматривайте сокрытие ссылок как меру безопасности.

Принудительное использование Flash для некоторых клиентов

По умолчанию Video.js использует Flash как обходное решение для браузеров, не поддерживающих элемент `video`. Возможно, в каких-то ситуациях вы захотите, чтобы некоторые браузеры использовали Flash даже при поддержке видео HTML5. В общем случае, если браузер пользователя поддерживает тег `<video>`, он не станет использовать Flash. Соответственно вам придется включить формат, поддерживаемый браузером, иначе обходное решение Flash не сработает. А если у вас уже имеется большой объем видеоматериалов в формате MP4 и заниматься их конвертацией нежелательно? Впрочем, Flash нормально воспроизводит файлы MP4.

Используя простую комбинацию проверки функциональности и параметров конфигурации Video.js, мы можем приказать Video.js использовать Flash, если браузер не поддерживает MP4.

После каждого тега `<video>` размещается следующий код, который проверяет, поддерживает ли браузер файлы MP4:

html5_video/mp4only_index.html

```
<script>
  var videoElement = document.createElement("video");
  if(videoElement.canPlayType){
    if (videoElement.canPlayType("video/mp4") === ""){
      videojs("video_blur", {"techOrder": ["flash","html5"]});
    };
  }
</script>
```

Мы создаем элемент `video` и проверяем, какие форматы он может воспроизводить, методом `canPlayType()`. Если метод `canPlayType()` возвращает пустую строку, значит, формат видео не поддерживается; в этом случае объект `videojs`, предоставляемый библиотекой Video.js, используется для настройки параметров проигрывателя.

Объект `videojs` получает в первом аргументе идентификатор тега `<video>`, за которым следует ассоциативный массив параметров, управляющих работой проигрывателя.

При таком решении браузер Firefox, не имеющий встроенной поддержки файлов MP4, переключается на Flash. Стоит заметить, что данное решение не оптимально; желательно закодировать видеоматериалы для поддерживаемых браузеров. Но если у вас нет такой возможности или ресурсов для ее реализации, это хороший компромисс.

Видео — прекрасный механизм распространения идей и информации, но что делать тем пользователям, которые не видят видео или не слышат аудио? Вы узнаете об этом в следующем рецепте!

Рецепт 22. Доступность видео

Ни одно из обходных решений не обладает особой эффективностью для пользователей с ограниченными возможностями. 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. Ниже приведен пример добавления простой расшифровки прямо под видеороликом.

html5_video/videojs_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. Если полная расшифровка невозможна, хотя бы создайте сводку с описанием важнейших частей видеосюжета. Даже это лучше, чем ничего.

Субтитры

Расшифровки полезны, но иногда пользователю нужно кое-что получше. HTML5 поддерживает возможность наложения субтитров с использованием нового тега `<track>`. Данная возможность еще не получила широкой встроенной поддержки, но в библиотеке Video.js она отлично поддерживается. Чтобы использовать ее, следует создать файл субтитров и хронометражных точек в формате Web VTT (Web Video Text Tracks). Формат VTT поддерживается в Internet Explorer 10, Chrome, Opera и Safari. Давайте опробуем его с одним из наших видеороликов.

Создайте папку `captions` в папке `video`. Затем создайте в этой папке файл *01_blur.vtt*. Файл содержит хронометражные точки и текст субтитров, которые должны выводиться в этих точках. На построение файла уходит некоторое время, ниже приводится окончательная версия.

html5_video/video/captions/01_blur.vtt

```
WEBVTT

00:00.000 --> 00:08.906
We'll drag the existing layer to the New button on the
bottom of the layers panel to create a new copy.

00:08.906 --> 00:14.167
Now we'll go to the Filter menu and choose Gaussian Blur.

00:14.167 --> 00:22.907
We'll change the blur amount just enough so we lose a
little detail of the image.

00:22.907 --> 00:33.670
Now we'll double-click on the layer to edit the layer.

00:33.670 --> 00:41.928
And we'll change the blending mode to overlay. This allows
the original layer underneath to show through.

00:41.928 --> 00:48.812
We can then adjust the amount of the effect by changing
the opacity.

00:48.812 --> 00:57.507
And now we have a slightly enhanced image.
```

Файл связывает субтитры с отдельными частями видео. Мы определяем временной интервал вывода и помещаем текст субтитров под диапазоном. Субтитры могут состоять из нескольких строк при условии, что в них нет пустых строк. Чтобы система работала с видео, мы добавляем в тег `<video>` тег `<track>`, который указывает на этот файл:

html5_video/videojs_index.html

```
<track kind="captions" src="video/captions/01_blur.vtt"
       srclang="en-us" label="English" />
```

Тег содержит информацию о том, что в файле определяются субтитры на английском языке. Также можно создать субтитры для других языков и сохранить их в отдельных файлах. Но для нашего примера этого достаточно; результат показан на рис. 24.

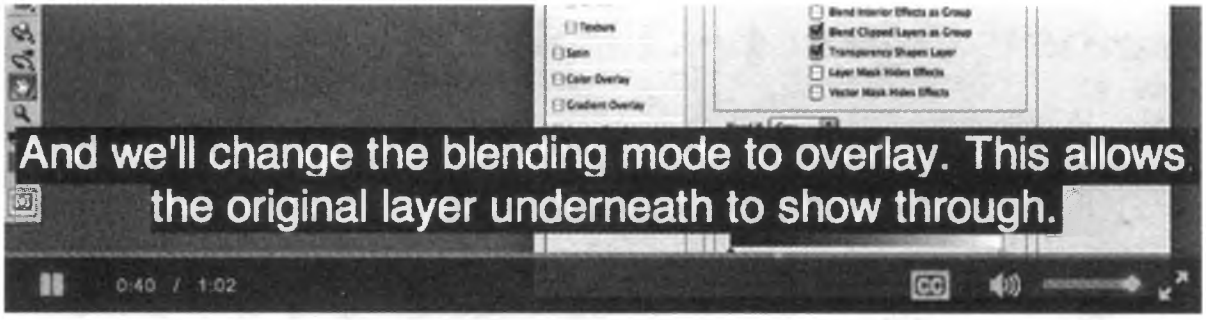


Рис. 24. Субтитры, отображаемые в Video.js

Построение субтитров для ваших видеоматериалов потребует значительных усилий, но они существенно упрощают жизнь пользователей с потерей слуха. Субтитры пригодятся и в тех ситуациях, когда пользователь вынужден отключать звук — например, при работе в офисе или библиотеке. А самое замечательное, что благодаря Video.js у вас имеется отличное обходное решение, которое вы можете использовать прямо сейчас.

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 плат-

формы 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 управления правами. Такие сайты, как Hulu, 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.

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 браузерами, а также управлять воспроизведением элементов audio.

В рецепте 20, «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:

html5_audio/javascripts/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.

Обязательно ознакомьтесь со всеми возможностями по спецификации[1].

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

Перспективы

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

Web VTT — еще одна область, заслуживающая пристального внимания. Возможности Web VTT не сводятся к простому созданию субтитров. Поддерживаются субтитры для нескольких языков, разбивка на главы для упрощения навигации и включение дополнительных метаданных о видеоматериале. Разработчик даже может использовать HTML и JSON для взаимодействия с видео, с выдачей событий при входе и выходе из хронометражных точек. Эта технология может стать отличной основой для построения интерактивных обучающих программ с встроенным видео. К сожалению, на момент написания книги не все браузеры обладали полной поддержкой API, а к тому времени, когда это произойдет, спецификация может измениться. И все же данная возможность определенно заслуживает внимания.

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. Далее будет рассмотрена новая возможность CSS3 `@font-face`, позволяющая использовать более симпатичные шрифты в блоге компании. В завершение главы вы увидите, как использовать CSS для выполнения анимации.

В этой главе рассматриваются следующие возможности CSS3:

`border-radius` [`border-radius: 10px;`]
: Закругление углов элементов. [C4, F3, S3.2, IE9, O10.5]

Поддержка RGBa [`background-color: rgba(255,0,0,0.5);`]
: Использование цветов RGB вместо шестнадцатеричных кодов (с альфа-каналом). [C4, F3.5, S3.2, IE9, O10.1]

`box-shadow` [`box-shadow: 10px 10px 5px #333;`]
: Создание теней, отбрасываемых элементами. [C3, F3.5, S3.2, IE9, O10.5]

Повороты [`transform: rotate(7.5deg);`]
: Поворот любого элемента. [C3, F3.5, S3.2, IE9, O10.5]

Градиенты [`linear-gradient(top, #fff, #efefef);`]
: Построение градиента для использования в качестве графического изображения. [C4, F3.5, S4]

`src: url(http://example.com/awesomeco.ttf); font-weight: bold; }`]
: Возможность использования конкретных шрифтов в CSS. [C4, F3.5, S3.2, IE5, O10.1]

Переходы [`transition: background 0.3s ease`]
: Плавный переход свойства CSS от одного значения к другому. [C4, F3.5, S4, IE10]

Анимации [`animation: shake 0.5s 1;`]
: Плавный переход свойства CSS от одного значения к другому посредством анимации по ключевым кадрам. [C4, F3.5, S4, IE10]

Рецепт 23. 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 Chrome, Firefox и Safari. В Internet Explorer 9 эта поддержка тоже появилась, так что теперь проблема решается без особых хлопот. Давайте посмотрим, как это делается.

Закругленные углы на форме входа

Вам поручено создать новую форму входа для сайта поддержки AwesomeCo. На полученных вами макетах изображены поля форм с закругленными углами. Начнем с реализации этого эффекта исключительно средствами CSS3. Примерный вид того, что нужно сделать, показан на рис. 25.

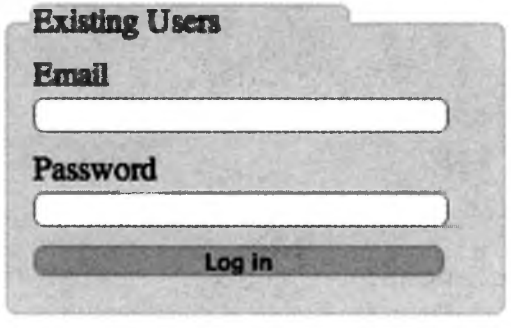


Рис. 25. Форма с закругленными углами

Форма входа в систему реализуется очень простым кодом HTML.

css3_rough_edges/index.html

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

Применение к форме стилей немного улучшит ее внешний вид.

css3_rough_edges/stylesheets/style.css

```
.login{
  width: 250px;
}

.login fieldset{
  background-color: #ddd;
  border: none;
}
.login legend{
  background-color: #ddd;
  padding: 0 64px 0 2px;
}

.login ol{list-style: none;
  margin: 2px;
  padding:0;
}
.login li{
  margin: 0 0 9px 0;
  padding: 0;
}

.login input{
  background-color: #fff;
  border: 1px solid #bbb;
  display:block;
  width: 200px;
}
.login input[type="submit"]{
  background-color: #bbb;
  padding: 0;
```

```
  width: 202px;
}
```

Эти стили удаляют из списка маркеры и обеспечивают равенство размеров текстовых полей. Также можно внести изменения в теги `<fieldset>` и `<legend>` формы для группировки полей. Когда базовое оформление будет завершено, можно переходить к применению эффектов закругления.

Для закругления углов всех текстовых полей формы понадобится правило CSS вида

css3_rough_edges/stylesheets/style.css

```
  .login input, .login fieldset, .login legend{
  border-radius: 5px;
}
```

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

Обходное решение

Приведенное решение работает в Firefox, Safari, Chrome и Internet Explorer 9 и 10, но не в Internet Explorer 8, а следовательно, нужно реализовать как можно более близкий аналог. Проблема совершенно тривиально решается при помощи библиотеки PIE[1], которая обеспечивает моментальную поддержку `border-radius` и ряда других функций. Загрузите PIE, распакуйте архив и поместите файл *PIE.htc* в папку *stylesheets*.

Прежде чем мы доберемся до закругления, необходимо избавиться от одной стилевой ошибки. Internet Explorer обрабатывает тег `<legend>` особым образом, так что при просмотре формы в IE мы не увидим заголовок группы на «корешке»; он полностью размещается внутри набора полей. Мы добавим небольшую «заплатку» для IE, которая сдвигает заголовок группы на несколько пикселов вверх — туда, где он выводится в Firefox и Chrome.

Создайте новый файл с именем *stylesheets/ie.css* и включите ссылку на него в условный комментарий, чтобы она загружалась только в IE 8 и ниже:

css3_rough_edges/index.html

```
<!--[if lte IE 8]>
<link rel="stylesheet" href="stylesheets/ie.css" type="text/css"
media="screen">
<![endif]-->
```

[1] http://css3pie.com/

Затем внесите в *stylesheets/ie.css* поправки для `legend` и `fieldset`:

css3_rough_edges/stylesheets/ie.css

```
.login {margin-top: 20px;}

.login fieldset legend{
  margin-top: -10px;
  margin-left: 10px;
}
.login fieldset{
  padding-left: 10px;
}
```

Группа полей смещается на 20 пикселов вниз, а заголовок группы — на 10 пикселов вверх. Также заголовок смещается вправо, но из-за особенностей стилевого оформления по умолчанию в Internet Explorer необходимо добавить небольшой отступ в саму группу полей. Теперь форма выглядит так же, как в других браузерах.

Наконец, мы загружаем PIE при помощи `behavior` — специального правила CSS, поддерживаемого Internet Explorer:

css3_rough_edges/stylesheets/ie.css

```
.login fieldset, .login input, .login legend{
  behavior: url(stylesheets/PIE.htc);
}
```

Обратите внимание: файл *PIE.htc* хранится в папке *stylesheets*. Ссылка на *PIE.htc* в нашей таблице стилей должна задаваться относительно страницы HTML, загружающей таблицу стилей, а не относительно самой таблицы.

Теперь во всех основных браузерах результат выглядит одинаково; версия для Internet Explorer показана на рис. 26.

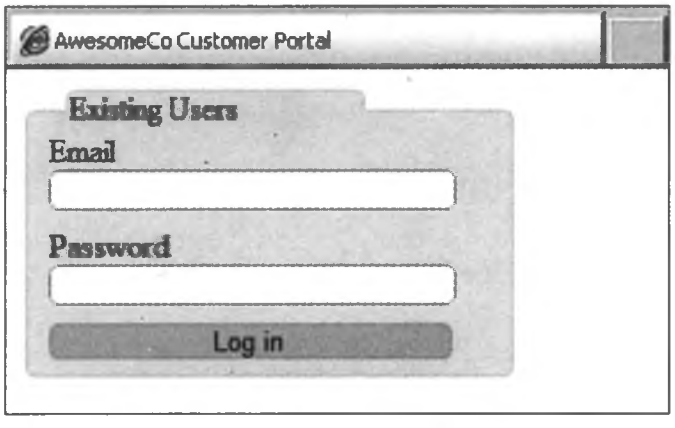


Рис. 26. Так форма будет выглядеть в Internet Explorer

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 используют Internet Explorer 9 и выше, возможно, вам просто не стоит тратить время на написание и сопровождение обходного решения.

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 24. 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. 27, со слегка наклоненной карточкой с именем и прозрачным белым пространством, в котором в будущем будет выводиться какой-то веб-контент.

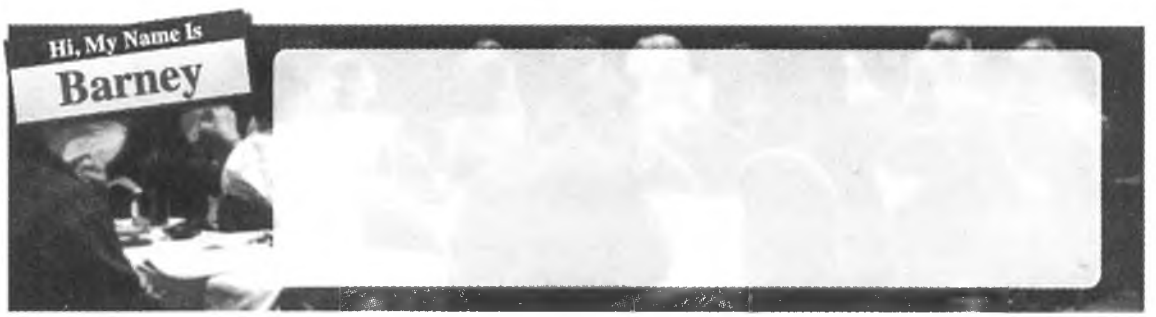


Рис. 27. Исходный макет, который мы воссоздадим средствами CSS3

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.

Базовая структура

Начнем с разметки базовой структуры страницы в HTML. Создайте файл HTML и включите в него следующий код:

css3_banner/index.html

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset="utf-8">
    <title>Sample Banner</title>
    <link rel="stylesheet" href="stylesheets/style.css">
```

```
    <div id="conference">
      <section id="badge">
        <h3>Hi, My Name Is</h3>
        <h2>Barney</h2>
      </section>

      <section id="info">
      </section>
    </div>

  </body>
</html>
```

Затем мы создадим файл *stylesheets/style.css* и добавим базовые стили для определения макета карточки и основной области контента.

css3_banner/stylesheets/style.css

```
#conference{
  background-color: #000;
  background-image: url('../images/awesomeconf.jpg');
  background-position: center;
  height: 240px;
  width: 960px;
}
#badge{
  border: 2px solid blue;
  display: block;
  text-align: center;
  width: 200px;
}
#info{
  display: block;
  height: 160px;
  margin: 20px;
  padding: 20px;
  width: 660px;
}
#badge, #info{
  background-color: #fff;
  float: left;
}
#badge h2{
  color: red;
```

```
  margin: 0;
  font-size: 40px;
}
#badge h3{
  background-color: blue;
  color: #fff;
  margin: 0;
}
```

После применения этой таблицы стилей к странице карточка с именем и область контента отображаются рядом друг с другом (рис. 28).

Рис. 28. Исходная версия баннера

Начнем со стилевого оформления карточки с именем.

Добавление градиента

Чтобы карточка с именем смотрелась более профессионально, мы заменим белый фон градиентом, переходящим от белого к светло-серому. Определяемый нами градиент станет фоновым изображением элемента. Градиенты работают в Firefox, Safari и Chrome, но реализация зависит от браузера.

В Firefox до версии 15 использовался метод `-moz-linear-gradient`, в параметрах которого указывается начальная точка градиента, начальный и конечный цвета. Браузеры на базе WebKit используют то же правило, но с заменой префикса `-moz-` на `-webkit-`.

Стандартный способ реализации градиентов базируется на методе `linear-gradient`[1] и выглядит практически так же, но использует направление `to bottom` вместо `top`. Поддерживаются направления `to bottom`, `to right`, `to left`, `to top`, а также конкретные углы (например, `45deg`).

[1] http://dev.w3.org/csswg/css3-images/#linear-gradients

Чтобы нужный эффект работал в наибольшем количестве браузеров, включите в таблицу стилей следующий фрагмент:

css3_banner/stylesheets/style.css

```
#badge{
  background-image: -webkit-linear-gradient(top, #fff, #eee);
  background-image:    -moz-linear-gradient(top, #fff, #eee);
  background-image:         linear-gradient(to bottom, #fff, #eee);
}
```

Для нашего примера этого достаточно, но также существует возможность применения радиальных градиентов и назначения промежуточных цветов. В нашем примере указаны только начальный и конечный цвета, 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. Свойство CSS3 `box-shadow` позволяет быстро определять тени для элементов[1].

Применение следующего правила к таблице стилей создает тень у карточки с именем. Фрагмент добавляется в селекторе `#banner`, непосредственно за только что определенным градиентом:

css3_banner/stylesheets/style.css

```
box-shadow: 5px 5px 5px 0px #333;
```

Свойство `box-shadow` имеет шесть параметров, хотя мы будем использовать только пять. 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. Третий параметр определяет радиус размывки. Со значением 0 тень выглядит очень контрастно, а с более высокими значениями она размывается. Четвертый параметр определяет ширину тени, а последний — ее цвет.

[1] http://www.w3.org/TR/css3-background/#the-box-shadow

Шестой параметр (`inset`), если он задан, размещает тень *внутри* поля. Таким образом, вместо используемой по умолчанию внешней тени создается внутренняя тень.

Обратите внимание на то, что в нашем примере не используются префиксы разработчиков. Internet Explorer 10, как и последние версии Chrome, Firefox, Safari и Opera, поддерживают эту функциональность без префиксов. Если вам потребуется обеспечить поддержку iOS 3, Android 2.1 или браузеров, выпущенных до 2011 года, используйте то же правило с префиксами `-moz-` и `-webkit-`.

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.

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 0px #bbbbbb;}
```

Вы указываете смещения X и Y, величину размывки, ширину и цвет тени.

Мы действуем так же, как при применении тени к другим элементам. Тени, отбрасываемые текстом, эффектно смотрятся, но при чрезмерной контрастности они затрудняют чтение. Прежде всего, следите за тем, чтобы ваш контент нормально читался.

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

[1] http://www.w3.org/TR/css3-2d-transforms/#transform-property

карточку, чтобы она выходила за границы контура баннера. Фрагмент, как и в предыдущем случае, добавляется в селектор `#banner`:

css3_banner/stylesheets/style.css

```
-webkit-transform: rotate(-7.5deg);
   -moz-transform: rotate(-7.5deg);
    -ms-transform: rotate(-7.5deg);
     -o-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. Впрочем, на этот раз стандартным правилом не обойтись — чтобы решение работало во всех браузерах, приходится применять префиксы.

Поворот выполняется так же просто, как и закругление углов, однако им 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!

Трансформации

Повороты — всего лишь одна из разновидностей трансформации элементов. Также можно выполнять трансформации масштабирования, сдвига и даже трехмерные трансформации. Существует еще более впечатляющая возможность: использование функции `matrix()` с `transform` позволяет еще точнее управлять преобразованием элементов. Для этого следует задать косинусы и синусы нужного угла. Допустим, мы хотим определить поворот карточки, используя `matrix()` вместо `rotate()`. Для этого следует взять используемый угол (−7,5°), вычислить и передать его косинус, отрицательное значение синуса, синус и снова косинус. Другими словами, мы определяем линейное преобразование по матрице 2 × 2:

```
-webkit-transform: matrix(0.99144,-0.13052,0.13052,0.99144,0,0);
   -moz-transform: matrix(0.99144,-0.13052,0.13052,0.99144,0px,0px);
    -ms-transform: matrix(0.99144,-0.13052,0.13052,0.99144,0,0);
     -o-transform: matrix(0.99144,-0.13052,0.13052,0.99144,0,0);
        transform: matrix(0.99144,-0.13052,0.13052,0.99144,0,0);
```

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.

С другой стороны, передавая нужные значения, вы сможете выполнять искажения, сдвиги, повороты и вообще любые трансформации. Освоив эту возможность, вы оцените ее мощь. Если вам захочется побольше узнать о трансформациях, обращайтесь по адресу *http://peterned.home.xs4all.nl/matrices/*.

Ох уж эта математика... Переходим к градиентам.

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.

css3_banner/stylesheets/style.css

```
#info{
  background-color: rgba(255,255,255,0.95);
  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 IE10 и других современных браузерах, но они не будут работать в Internet Explorer 8 и 9. Их можно в определенной степени эмулировать через фильтры Microsoft DirectX, но фильтры поглощают слишком много вычислительных ресурсов и нарушают работу других функций, а пользовательский интерфейс получается просто ужасным. Поверьте, в данной ситуации лучше обойтись без обходного решения. Помните, что все возможности этого раздела относятся исключительно к представлению информацию. Браузеры, не поддерживающие синтаксис CSS3, все равно смогут отображать контент так, чтобы он нормально читался.

Конечно, если вы любознательны и склонны к самостоятельным исследованиям, ознакомьтесь с файлом *css3_banner/stylesheets/ie.css* из архива исходного кода книги — он пытается организовать работу описанных эффектов в Internet Explorer 8 и ниже. Но учтите — это решение не идеально, а его синтаксис достаточно сложен. А самое неприятное, что конечный результат выглядит убого.

Конечно, вы можете использовать условную таблицу стилей для применения фона PNG к контейнерному элементу. Но спросите себя, стоит ли это делать? Трансформации, градиенты и тени — все это, конечно, хорошо, но люди посещают веб-страницы для того, чтобы читать контент. Правильный выбор шрифта может существенно изменить впечатления пользователя, а CSS3 открывает для нас новые возможности для управления шрифтами. Давайте посмотрим, как это делается.

Рецепт 25. Использование шрифтов

Подбор шрифтов оказывает серьезное влияние на впечатления 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

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», чтобы все желающие смогли увидеть, как выглядит этот шрифт на практике. Итак, давайте посмотрим, как выбрать и использовать шрифт в «чистом» CSS.

@font-face

Вообще говоря, директива `@font-face` появилась в спецификации CSS2 и была реализована еще в Internet Explorer 5[1]. Однако в реализации Microsoft использовался формат шрифтов EOT (Embedded OpenType), а большинство современных шрифтов хранится в формате TrueType или OpenType, которые поддерживаются всеми основными браузерами.

Форматы шрифтов

Шрифты существуют во многих форматах. Выбор формата, который должен поставляться посетителям сайта, зависит от браузера.

Тип шрифта	Поддерживаемые браузеры
WOFF (Web Open Font)	[F3.6, C5, S5.1, IE9, O11.1, iOS5]
TrueType (TTF)	[F3.5, C4, S3, O10, iOS4.2, A2.2]
OpenType (OTF)	[F3.5, C4, S3, O10, iOS4.2, A2.2]
Embedded OpenType (EOT)	[IE5–8]
Scalable Vector Graphics (SVG)	[iOS]

Microsoft, Opera и Mozilla совместно создали формат Web Open Font, поддерживающий сжатие без потерь и расширенные возможности лицензирования для создателей шрифтов.

Чтобы ваше решение работало во всех браузерах, шрифты придется публиковать в нескольких форматах. Посмотрим, как это делается.

Смена шрифта

Нужный нам шрифт доступен на сайте FontSquirrel[2] в форматах TrueType, WOFF, SVG и EOT — как раз то, что нужно.

---

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

[2] Вы можете загрузить его с сайта http://www.fontsquirrel.com/fonts/Garogier или взять из примеров кода книги.

Использование шрифта состоит из двух шагов: определения шрифта и присоединения его к элементам. Включите в таблицу стилей блога следующий код:

css3_fonts/stylesheets/style.css

```
@font-face {
  font-family: 'GarogierRegular';
  src: url('fonts/garogier_unhinted-webfont.eot?#iefix')
       format('embedded-opentype'),
       url('fonts/garogier_unhinted-webfont.woff') format('woff'),
      url('fonts/garogier_unhinted-webfont.ttf') format('truetype'),
       url('fonts/garogier_unhinted-webfont.svg#garogierregular')
       format('svg');
  font-weight: normal;
  font-style: normal;
}
```

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.

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

[2] http://code.google.com/apis/webfonts/

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.

Префикс `?#iefix'` в файле *.eot* устраняет ошибку разбора в Internet Explorer 8. Без этой разметки IE8 выдает ошибки 404 для остальных правил. Вопросительный знак заставляет IE8 считать, что после EOT следуют параметры запроса, так что фактически остаток игнорируется.

Предполагается, что все шрифты помещены в папку *stylesheets/fonts*. Ссылки на шрифты должны задаваться относительно местонахождения таблицы стилей, а не страницы HTML, в которой эта таблица стилей используется. И это вполне разумно, потому что на реальном сайте эта таблица стилей будет вызываться из разных страниц HTML.

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.

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. 29.

В современных браузерах изменить шрифт относительно несложно, но мы также должны позаботиться о браузерах, в которых такая возможность пока не поддерживается.

[1] http://www.fontsquirrel.com/fontface/generator

Рис. 29. Блог с новым шрифтом

Обходное решение

Обходные решения для разных версий Internet Explorer и других браузеров уже были представлены ранее, однако также необходимо позаботиться о том, чтобы страницы нормально читались в браузерах без поддержки `@font-face` (или если браузеры по какой-то причине не смогли загрузить наш шрифт).

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:

css3_fonts/stylesheets/style.css

```
font-family: "GarogierRegular", Georgia,
             "Palatino", "Palatino Linotype",
             "Times", "Times New Roman", serif;
```

[1] http://unitinteractive.com/blog/2008/06/26/better-css-font-stacks/

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.

А теперь посмотрим, как использовать CSS для создания анимаций.

Рецепт 26. Переходы и анимации

В CSS3 существует два способа «оживления» контента: *переходы* и *анимации*. Они работают сходным образом, но служат двум разным целям. Переход указывает, что свойство должно плавно изменяться от одного значения до другого. Анимация позволяет действовать более точно, определяя ключевые кадры для сложных анимаций.

Нам поручено «украсить» форму входа в систему, созданную в рецепте 23. Руководитель продукта хочет, чтобы при получении фокуса поля формы плавно окрашивались в другой цвет, а при вводе неверного имени пользователя и пароля форма «вибрировала». Для полей формы можно использовать простые переходы, а для формы — анимацию.

Реализация эффекта растворения

В рецепте 9 мы написали код CSS, который изменял цвет фона элемента при получении им фокуса:

```
li>span[contenteditable=true]:focus{
  background-color: #ffa;
  border: 1px shaded #000;
}
```

В этом случае изменение происходит резко, а новый цвет фона и границы просто заменяют старые. С переходами CSS мы можем продлить их по времени. Для этого нужно лишь определить, как должен происходить переход, сколько времени он должен занимать и какие свойства в нем должны быть задействованы.

В определении перехода могут использоваться следующие свойства:

- `transition-property` — свойство CSS, к которому применяется переход.
- `transition-duration` — время, за которое должен выполняться переход.
- `transition-delay` — задержка перед началом перехода.
- `transition-timing-function` — способ задания промежуточных значений перехода.

Функции плавности

Помните, как на уроках алгебры учитель заставлял вас рисовать графики функций, а вы никак не могли понять, для чего это нужно?

Свойство `transition-timing-function` описывает закономерность изменения свойства по времени на протяжении анимации. Функция плавности задается кубической кривой Безье, которая определяется четырьмя

контрольными точками. Каждая точка характеризуется координатами x и y в диапазоне от 0 до 1. Первая и последняя контрольные точки обычно имеют координаты (0.0, 0.0) и (1.0, 1.0), а две средних определяют форму кривой.

В спецификации определено несколько встроенных функций плавности:

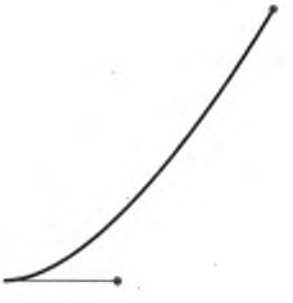
- `linear`
- `ease-in`
- `ease-out`
- `ease-in-out`
- `ease`

Если изменение должно выполняться с постоянной скоростью, используется функция `linear`. Если оно должна начинаться медленно и постепенно ускоряться, используется функция `ease-in`. Чтобы оно начиналось медленно, постепенно ускорялось, а потом замедлялось, используется функция `ease-out`.

Каждая из этих функций определяет кубическую кривую Безье. И хотя в большинстве случаев этих функций оказывается достаточно, если вы будете понимать, как они работают, то сможете определять собственные функции плавности по четырем точкам на графике при помощи функции `cubic-bezier`. Рассмотрим несколько примеров.

У линейной функции плавности контрольные точки совпадают с двумя конечными, в результате чего определяется прямая линия, наклоненная под углом в 45°. Четыре точки функции `linear` имеют координаты ((0.0, 0.0), (0.0,0.0), (1.0, 1.0), (1.0, 1.0)), а сама функция выглядит так:

Более сложная кривая с точками ((0.0, 0.0), (0.42,0.0), (1.0, 1.0), (1.0, 1.0)), соответствующая функции `ease-in`, выглядит так:

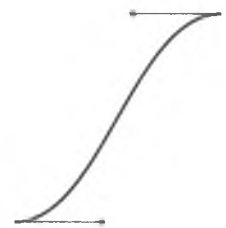


На этот раз изменилась только вторая точка; это и привело к искривлению левой нижней части кривой. Таким образом, анимация начинается медленно, а потом постепенно ускоряется.

Сравните эту функцию с функцией `ease-out`, определяющей анимацию, которая начинается с постоянной скоростью и замедляется к концу:

Кривая определяется контрольными точками ((0.0, 0.0), (0.0,0.0), (0.58, 1.0), (1.0, 1.0)).

У функции `ease-in-out` искривлены как нижняя, так и верхняя часть:



Кривая определяется контрольными точками ((0.0, 0.0), (0.42,0.0), (0.58, 1.0), (1.0, 1.0)), анимация ускоряется в начале и замедляется к концу.

Функция `ease` похожа на `ease-in-out`, но в начале анимация выполняется чуть быстрее, чем в конце.

Вы также можете определить собственную функцию плавности, передавая четыре контрольные точки функции при вызове функции `cubic-bezier()`:

css3_animation/examples/style.css

```
.bounce{
  transition-property: left;
  transition-timing-function: cubic-bezier(0.1, -0.6, 0.2, 0);
  transition-duration: 1s;
}
.bounce:hover{
  left: 200px;
}
```

Эта функция плавности создает небольшой эффект «рикошета» из-за отрицательного значения первой контрольной точки (начальная точка по-прежнему имеет координаты (0.0, 0.0)).

Если вы захотите больше узнать о создании таких кривых, следующий сценарий поможет вам поближе познакомиться с примерами и узнать конкретные значения координат: *http://www.netzgesta.de/dev/cubic-bezier-timing-function.html.*

Создание собственных переходов

При передаче фокуса поля должен происходить переход цвета. Для этого мы определяем для элементов формы свойства `transition`. Считайте это своего рода первой фазой перехода. Браузер узнает, что он должен отслеживать изменения этих свойств и как должны анимироваться их изменения.

```
input[type="email"], input[type="password"]{
  transition-timing-function: linear;
  transition-property: background, border;
  transition-duration: 0.3s;
}
```

Это стандартный способ определения переходов, но если вы хотите, чтобы ваше решение работало во всех браузерах, вам придется повторно определить эти переходы с префиксами `-webkit-` и `-moz-`, как это делалось с другими свойствами CSS. Запись получается довольно длинной. К счастью, существует сокращенный вариант, рекомендованный для определения переходов:

css3_animation/stylesheets/style.css

```
.login input[type="email"], .login input[type="password"]{
  -webkit-transition: background 0.3s linear
                      border 0.3s linear;
     -moz-transition: background 0.3s linear,
                      border 0.3s linear;
       -o-transition: background 0.3s linear,
                      border 0.3s linear;
          transition: background 0.3s linear,
                      border 0.3s linear;
}
```

Сокращенная запись `transition` позволяет передать CSS свойство, к которому применяется переход, продолжительность и функцию плавности.

Вы можете задать несколько свойств, продолжительностей и функций плавности, разделяя их запятыми.

И конечно, для поддержки старых версий Firefox и Opera вам придется снова определить их с префиксами `-moz-` и `-o-` соответственно. Для экономии места их запись здесь не приводится.

Итак, переходы определены, и теперь при добавлении эффектов с использованием синтаксиса `:focus`:

css3_animation/stylesheets/style.css

```
.login input[type="email"]:focus, .login
input[type="password"]:focus{
  background-color: #ffe;
  border: 1px solid #0e0;
}
```

браузер реализует плавный переход фона и границы.

Переходы обеспечивают простейший, но не единственный механизм плавного изменения свойств CSS от одного значения к другому.

Эффект вибрации на базе анимации CSS

Переходы отлично подходят для интерполяции изменения свойств от одного состояния к другому. Но чтобы создать эффект покачивания из стороны в сторону, потребуется нечто более мощное — например, анимация CSS с *ключевыми кадрами*[1]. Вибрация представляет собой не что иное, как многократное перемещение поля влево и вправо.

Давайте определим анимацию для эффекта вибрации. В файле *stylesheets/style.css* следует определить ключевые кадры с использованием синтаксиса `@keyframes`:

css3_animation/stylesheets/style.css

```
@keyframes shake{
  0%{left:0;}
  20%{left:-2%;}
  40%{left:2%;}
  60%{left:-2%;}
  80%{left:2%;}
  100%{left:0;}
}
```

Это стандартное решение работает в Internet Explorer 10, а также последних версиях Firefox и Chrome, но если вы хотите, чтобы оно работало

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

и в Safari, придется снова определить ключевые кадры с префиксом браузера, как и в случае с переходами.

css3_animation/stylesheets/style.css

```
@-webkit-keyframes shake{
  0%{left:0;}
  20%{left:-2%;}
  40%{left:2%;}
  60%{left:-2%;}
  80%{left:2%;}
  100%{left:0;}
}
```

Не забудьте о префиксах `-moz-` и `-opera-`, если вы хотите поддерживать эти браузеры.

После определения ключевых кадров анимации можно применить анимацию к правилу CSS. Эффект вибрации должен срабатывать только тогда, когда пользователь отправляет форму с неверно заданным именем пользователя и паролем. Мы используем jQuery для перехвата события отправки формы и выполнения вызова Ajax. В форму будет добавлен класс `shake`, который запустит анимацию. Начнем с определения правила `shake` в таблице стилей:

```
.shake{
  animation-name: shake;
  animation-duration: 0.5s;
  animation-delay: 0;
  animation-iteration-count: 1;
  animation-timing-function: linear;
}
```

Эта разметка очень похожа на определения переходов из предыдущего рецепта. Вы можете управлять анимацией, функцией плавности, продолжительностью, задержкой и количеством итераций.

Определение получается довольно объемистым, особенно если понадобится добавить к этим правилам префиксы браузеров. Используем сокращенную запись:

```
css3_animation/stylesheets/style.css
  .shake{
    -webkit-animation: shake 0.5s 1;
       -moz-animation: shake 0.5s 1;
            animation: shake 0.5s 1;
}
```

Часть, относящаяся к CSS, на этом завершена. Остается лишь применить стиль при неудачной отправке данных формы. Сначала создадим функцию, выполняющую запрос Ajax. Включите в файл *javascripts/form.js* новый метод:

css3_animation/javascripts/form.js

```
var processLogin = function(form, event){
  event.preventDefault();
  var request = $.ajax({
    url: "/login",
    type: "POST",
    data: form.serialize(),
    dataType: "json"
  });
  request.done = function(){
    // Действия, выполняемые при удачном входе
  };
  return(request);
};
```

Функция получает два параметра: объект jQuery с отправляемой формой и событие. Мы блокируем поведение события по умолчанию и строим запрос Ajax, передавая ему сериализованную форму в качестве данных. Для определения того, что должно произойти при получении успешного ответа от сервера, используется поддержка *обязательств* (promises) jQuery. Обязательства позволяют писать асинхронный код без использования вложенных обратных вызовов.

Метод jQuery `$.ajax()` возвращает объект, реализующий интерфейс Promise. Вместо определения обратного вызова `success()` в методе `$.ajax()` мы можем определить обратные вызовы `done()` и `fail()` для объекта, возвращаемого `$.ajax()`. Эти обратные вызовы выполняются при завершении запроса Ajax или немедленно, если запрос уже завершен. Это позволяет нам определять обратные вызовы на программном уровне даже в других частях программы. Кроме того, код при этом разбивается на меньшие логические фрагменты (вместо одной большой кучи неконтролируемых обратных вызовов). За более подробной информацией об обязательствах обращайтесь к книге Тревора Бернэма «Async JavaScript: Build More Responsive Apps with Less Code» [Bur12].

Обратный вызов `done()` уже определен в этой функции `processLogin()`. Чтобы форма вибрировала при неудачном входе, необходимо создать двух слушателей событий. Первый слушатель обрабатывает событие отправки

Обходное решение

Чтобы обеспечить работу всех этих переходов и анимаций, проще всего воспользоваться обходным решением на базе jQuery. Нам понадобится как jQuery, так и плагин jQuery Color, а поскольку в разных браузерах используется разное поведение, мы также используем Modernizr для обнаружения поддержки и активизации соответствующего кода.

Начнем с добавления Modernizr в секцию `<head>` страницы. При этом используется версия с `load()`, приведенная в разделе «Modernizr» рецепта 5.

css3_animation/index.html

```
<script src="javascripts/modernizr.js"></script>
```

Загрузите плагин jQuery Color и поместите его в папку *javascripts*. Этот плагин необходим для применения анимации к цветам[1].

Теперь можно переходить непосредственно к обходным решениям.

Реализация переходов средствами jQuery

Эффект цветового перехода, который должен воспроизводиться при выделении текстового поля, реализуется при помощи метода jQuery `animate()`. Для этого понадобятся два события; при получении фокуса поле должно окрашиваться в желтый цвет, а при потере — возвращаться обратно к белому. Вот как это делается:

css3_animation/javascripts/form.js

```
var addTransitionFallbackListeners = function(){
  $(".login input[type=email],
     .login input[type=password]").focus(function(){
    $(this).animate({
         backgroundColor: "#ffe"
    }, 300 );
  });
  $(".login input[type=email], .login input[type=password]").
blur(function(){
    $(this).animate({
         backgroundColor: "#fff"
    }, 300 );
  });
};
```

Мы определяем эти обратные вызовы внутри функции, после чего используем Modernizr для проверки поддержки переходов, загружаем

[1] http://code.jquery .com/color/jquery .color-2.1.2.min.js

плагин jQuery Color в случае ее отсутствия, а затем вызываем функцию, определяющую обратные вызовы:

css3_animation/javascripts/form.js

```
Modernizr.load(
  {
    test: Modernizr.csstransitions,
    nope: "javascripts/jquery.color-2.1.2.min.js",
    callback: function(url, result){
      if (!result){
        addTransitionFallbackListeners();
      }
    }
  }
);
```

Этого достаточно для того, чтобы переходы заработали. Как видите, с jQuery поддержка переходов реализуется достаточно просто. Теперь займемся анимациями.

Реализация анимаций средствами jQuery

Для создания эффекта вибрации формы мы воспользуемся функцией jQuery `animate()`. В файле *javascripts/form.js* определяется новая функция с именем `addFormSubmitWithFallback()`, которая обрабатывает отправку формы, вызывает существующий метод `processLogin()` и определяет обратный вызов `fail()` — все делается так же, как прежде, но на этот раз для анимации применяется jQuery.

css3_animation/javascripts/form.js

```
var addFormSubmitWithFallback = function(){
  $(".login").submit(function(event){
    var form = $(this);
    request = processLogin(form, event);
    request.fail(function(){
      form.animate({left: "-2%"}, 100)
        .animate({left: "2%"}, 100)
        .animate({left: "-2%"}, 100)
        .animate({left: "2%"}, 100)
        .animate({left: "0%"}, 100);
    });
  });
};
```

Наконец, мы используем Modernizr для проверки поддержки анимации. Если браузер поддерживает анимацию, то используется исходное решение. Если анимация не поддерживается, вызывается обходное решение, которое вызывает исходный код обработки формы, но на этот раз использует обратный вызов `fail()` для присоединения анимации.

css3_animation/javascripts/form.js

```
if(Modernizr.cssanimations){

  addFormSubmitWithCSSAnimation();
  addAnimationEndListener();

}else{
addFormSubmitWithFallback();
}
```

Код в определенной степени дублируется, но вынесение большей части общей функциональности в отдельную функцию позволяет нам контролировать ситуацию (а заодно кое-что узнать об обязательствах).

Ненадолго задумайтесь над тем, что мы здесь сделали, и задайте себе вопрос — а так ли это необходимо? Как и в других темах этой главы, может оказаться, что создание обходного решения не окупает потраченного времени. Допустим, элемент не вибрирует у 15° пользователей — и что? Если вы *можете* добавить обходное решение, это не значит, что его *нужно* добавлять — если, конечно, от этого не зависит ваша следующая зарплата.

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, отражения и даже эффекты фильтров.

Модули CSS3 (когда работа над ними будет завершена) значительно упростят создание полнофункциональных, более совершенных и внешне привлекательных интерфейсных элементов. То, что вы привыкли делать в jQuery — от создания простых меню до сложных графических элементов, — можно будет делать полностью средствами CSS. Следите за спецификациями и продолжайте эксперименты!

III

За пределами разметки

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. Вы можете манипулировать с историей просмотра, создавать более интерактивные интерфейсы и устанавливать долгосрочные подключения к серверу.

Некоторые из этих возможностей появились на базе спецификации HTML5. Другие никогда в эту спецификацию не входили, но создатели браузеров и разработчики связывают их с HTML5, потому что спецификация реализуется в них наряду с другими возможностями. Как бы то ни было, это полезные инструменты, которые помогают нам сделать работу пользователя более приятной.

9 Хранение данных на стороне клиента

Помните, как нам нравилось работать с cookie? Нет? Я тоже. 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 часто тем или иным образом отключают cookie.

В HTML5 появились новые технологии хранения данных на стороне клиента: Web Storage (с использованием `localStorage` или `sessionStorage`)[1], IndexedDB[2] и Web SQL Databases[3]. Они обладают чрезвычайно широкими возможностями и в достаточной степени безопасны. И что самое лучшее, они уже реализованы в нескольких браузерах, включая Safari для iOS и браузер Android 2.0. Тем не менее формально эти технологии не являются частью спецификации HTML5, — они были выделены в отдельные спецификации.

Хотя эти механизмы 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 или выполняя синхронизацию 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, от настольных компьютеров до планшетов и смартфонов. Вы научитесь использовать эти технологии для сохранения пользовательских настроек и создания простой базы данных.

В этой главе рассматриваются следующие возможности:

`localStorage`

: Хранение данных в виде пар «ключ/значение» с привязкой к домену и сохранением данных между сеансами. [C5, F3.5, S4, IE8, O10.5, IOS3.2, A2.1]

`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, IOS3.2, A2.1]

---

[1] http://www.w3.org/TR/webstorage/

[2] http://www.w3.org/TR/IndexedDB

[3] http://www.w3.org/TR/webdatabase/

IndexedDB

Хранилище объектов, содержимое которого сохраняется между сеансами. [C25, F10, IE10]

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. Технология исключена из активной спецификации. [C5, S3.2, O10.5, IOS3.2, A2]

Offline Web Applications

Кэширование файлов для автономного использования; позволяет приложениям работать без подключения к Интернету. [C4, F3.5, S4, O10.6, IOS3.2, A2]

Рецепт 27. Сохранение настроек с использованием Web Storage

Механизм Web Storage 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. Сохранение и чтение строк данных легко реализуется минимальным объемом кода JavaScript. Это один из самых распространенных интерфейсов хранения информации, реализованный в Internet Explorer 8, а также в старых версиях iOS и браузера Android.

Информация, хранимая в подсистеме `localStorage` механизма Web Storage, сохраняется между сеансами и не может читаться другими сайтами, потому что доступ к ней ограничивается текущим посещаемым доменом. Будьте внимательны при локальной разработке. Если вы работаете на локальном сервере (например, `localhost`), в переменных легко возникает путаница, и вам придется постоянно приказывать браузеру очистить хранилище.

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 Web Storage, чтобы сохраненные данные не терялись между сеансами. После завершения работы у нас должен получиться прототип, показанный на рис. 30.

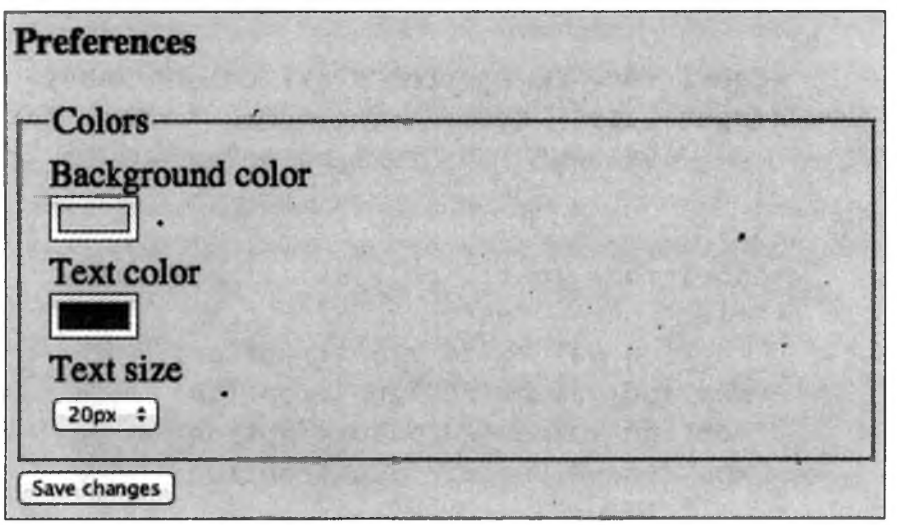


Рис. 30. Страница настроек

Построение формы

При построении формы ввода настроек будет использоваться семантическая разметка HTML5 и новые элементы, описанные в главе 3. Поль-

зователю предоставляется возможность выбрать цвет текста, цвет фона, а также размер шрифта.

html5_localstorage/index.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>Preferences</title>
    <link rel="stylesheet" href="stylesheets/style.css">
    <script src="javascripts/modernizr.js"></script>
  </head>
  <body>
    <div id="container">
      <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 class="color" type="color"
                      name="background_color"
                     value="" id="background_color">
            </li>
            <li>
              <label for="text_color">Text color</label>
              <input class="color" 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>
    </div>
```

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

Для представления цветов используются коды HTML, а для полей — тип HTML5 `color`. Также будет добавлен небольшой фрагмент CSS для стилевого оформления формы.

html5_localstorage/stylesheets/style.css

```
form ol{
  list-style: none;
  margin: 0;
  padding: 0;
}

form li{
  margin: 0;
  padding: 0;
}

form li label{ display:block; }
```

Прототип готов к использованию. Теперь займемся сохранением изменений, внесенных на форме.

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/javascripts/storage.js

```
localStorage.setItem("background_color", $("#background_color").
val());
```

Не сложнее выполняется и чтение сохраненных данных.

html5_localstorage/javascripts/storage.js

```
var bgcolor = localStorage.getItem("background_color");
```

Напишем вспомогательный метод для сохранения всех настроек с формы. Создайте файл *javascripts/storage.js* и добавьте в него следующий код:

html5_localstorage/javascripts/storage.js

```
var save_settings = function(){
  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();
};
```

Метод извлекает значения из полей формы и помещает их в ключи `localStorage`.

Аналогичный метод загружает данные из системы `localStorage` и заполняет ими поля формы.

html5_localstorage/javascripts/storage.js

```
var load_settings = function(){
  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 разделе.

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/javascripts/storage.js

```
var apply_preferences_to_page = function(){
  $("body").css("backgroundColor", $("#background_color").val());
  $("body").css("color", $("#text_color").val());
  $("body").css("fontSize", $("#text_size").val() + "px");
};
```

Наконец, вся эта логика активизируется в конце документа.

html5_localstorage/javascripts/storage.js

```
load_settings();

$('form#preferences').submit(function(event){
  event.preventDefault();
  save_settings();
});
```

Последнее, что осталось сделать — загрузка сценария и jQuery при загрузке страницы HTML. Включите следующие две строки непосредственно перед закрывающим тегом <body>:

html5_localstorage/index.html

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

Теперь данные сохраняются между сеансами браузера.

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');
```

Такое решение намного удобнее традиционных решений с cookie.

Обходное решение

Метод `localStorage` работает во всех современных браузерах и нескольких старых, так что обходное решение на стороне клиента не понадобится. Однако данными, хранимыми в `localStorage`, не удастся обменяться с сервером или другим компьютером, так что если пользователь дома изменит конфигурацию, эти изменения не будут доступны на его рабочих компьютерах. Кроме того, решение не работает, если пользователь отключил JavaScript. Таким образом, хорошее обходное решение будет

основано на сохранении данных на сервере. В таких приложениях при нажатии кнопки Submit данные могут сохраняться на сервере, связываясь с записью пользователя. Форма строится так, что она отправляет данные непосредственно на сервере, а в страницу вносятся изменения: при отсутствии настроек на стороне клиента используются настройки со стороны сервера. К сожалению, написание серверного решения выходит за рамки книги.

Технология Web Storage хорошо подходит для малых блоков данных, но во-первых, ее производительность на мобильных устройствах оставляет желать лучшего, а во-вторых, она не предназначена для больших объемов данных. Рассмотрим возможности хранения более сложных структур данных на стороне клиента.

Рецепт 28. Хранение информации в базе данных на стороне клиента с использованием IndexedDB

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. К сожалению, от этой спецификации отказались в пользу IndexedDB — технологии невероятно мощной, хотя и более сложной в использовании. Рассмотрим использование IndexedDB на примере простого клиентского веб-приложения. Прежде чем браться за дел ›, следует сказать, что в этом рецепте используется большой объем кода JavaScript, и он получился довольно длинным, но гибкость механизма хранения данных на стороне клиента оправдывает затраченные усилия..

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.

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. 31.

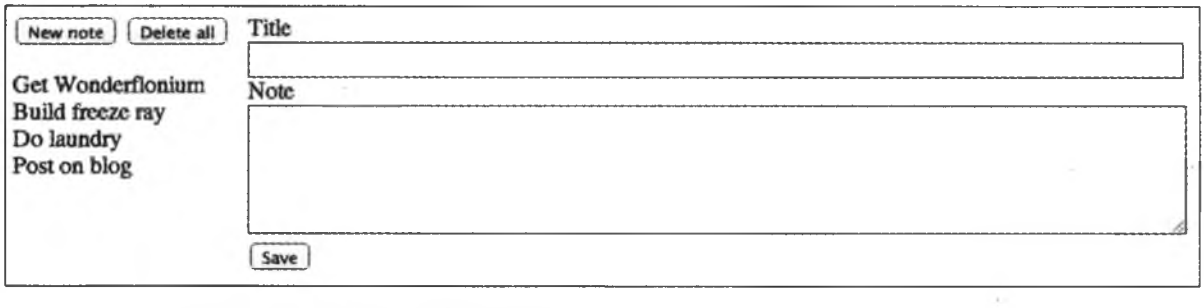


Рис. 31. Интерфейс приложения для работы с заметками

[1] http://dev.w3.org/html5/webdatabase/

Начнем с разметки пользовательского интерфейса.

html5_indexedDB/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>AwesomeNotes</title>
    <link rel="stylesheet" href="stylesheets/style.css">
    <script src="javascripts/IndexedDBShim.min.js"></script>
  </head>
  <body>
    <div id="container">
      <section id="sidebar">
        <input type="button" id="new_button" value="New note">
        <input type="button" id="delete_all_button"
               value="Delete all">
        <ul aria-live="polite" id="notes">
        </ul>
      </section>
      <section id="main" aria-live="polite">
        <form>
          <ol>
            <li>
              <label for="title">Title</label>
              <input type="text" id="title">
            </li>
            <li>
              <label for="note">Note</label>
              <textarea id="note"></textarea>
            </li>
            <li>
              <input type="submit" id="save_button" value="Save">
              <input type="submit" id="delete_button"
                     value="Delete">
            </li>
          </ol>
        </form>
      </section>
    </div>
  </body>
</html>
```

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. 31, мы применим к нему стилевое оформление. Файл *style.css* выглядит так:

html5_indexedDB/stylesheets/style.css

```
#container{
  margin: 0 auto;
  width: 80%;
}
#sidebar, #main{
  display: block;
  float: left;
}

#main{ width: 80%; }

#sidebar{ width: 20%;}

form ol{
  list-style: none;
  margin: 0;
  padding: 0;
}

form li, #sidebar li{
  margin: 0;
  padding: 0;
}

form li label{ display:block; }

#note, #title{
  border: 1px solid #000;
  font-size: 20px;
  width: 100%;
}

#sidebar ul{
  list-style: none;
  padding: 0;
};
```

```
#sidebar li{ cursor: hand; cursor: pointer; }

#title{ height: 20px; }

#note{ height: 80px; }
```

Таблица стилей отключает маркеры элементов списка, 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

Создайте файл *javascripts/notes.js*. В этом файле будет размещаться вся логика приложения. Подключите его в нижней части *index.html*, непосредственно над тегом `<body>` вместе с библиотекой jQuery, которая будет использоваться для обработки событий и упрощения манипуляций с моделью DOM.

html5_indexedDB/index.html

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

В файле *javascripts/notes.js* необходимо определить несколько переменных для нашего приложения.

html5_indexedDB/javascripts/notes.js

```
// Ссылка на базу данных
var db = null;
window.indexedDB = window.indexedDB || window.mozIndexedDB ||
                   window.webkitIndexedDB ||window.msIndexedDB;
```

В начале сценария объявляется переменная `db`. Через эту переменную с базой данных будут работать остальные созданные нами методы. Переменная объявляется в глобальной области видимости, что вообще-то делать не рекомендуется. Я решил по возможности упростить код JavaScript в нашем примере.

Затем мы определяем переменную `window.indexedDB`, потому что, к сожалению, в разных браузерах используются разные имена этого объекта.

К счастью, несмотря на различия в именах, они работают практически одинаково, так что мы можем создать одну переменную и сослаться на нее.

Создадим простую функцию, которая обеспечивает подключение к базе данных:

html5_indexedDB/javascripts/notes.js

```
var connectToDB = function(){
  var version = 1;
  var request = window.indexedDB.open("awesomenotes", version);
  request.onsuccess = function(event) {
    db = event.target.result;
    fetchNotes();
  };
  request.onerror = function(event){
    alert(event.debug[1].message);
  }
};
```

Для создания подключения используется метод `open()` объекта `indexedDB`. При вызове передается версия схемы, а возвращается объект `request`. Объект `request` содержит метод обратного вызова `success()`, который срабатывает при успешном подключении, и метод `error()`, который срабатывает при неудачной попытке.

В методе `success()` вызывается метод `fetchNotes()`, который мы построим позже. Этот метод обращается к базе данных и загружает заметки на боковую панель. Но прежде чем выполнять какие-либо операции с базой данных, ее необходимо создать.

Создание таблицы

Таблица для хранения заметок состоит из трех столбцов.

Столбец	Описание
id	Уникальный идентификатор поля
title	Название заметки
Note	Текст заметки

Для создания таблицы мы подключаемся к методу обратного вызова объекта `request`, созданного в только что определенном методе `connectToDB()`. Обратите внимание: `onupgradeneeded()` срабатывает при изменении номера версии схемы. В нашем случае метод `connectToDB()` задает версию и подключается к еще несуществующей базе данных. Брау-

зер инициирует обратный вызов onupgradeneeded(), чтобы мы получили возможность создать таблицу. Добавьте следующий код в connectToDB():

html5_indexedDB/javascripts/notes.js

```
var connectToDB = function(){
  var version = 1;
  var request = window.indexedDB.open("awesomenotes", version);

➤ request.onupgradeneeded = function(event) {
➤ alert("unupgradeneeded fired");
➤ var db = event.target.result;
➤ db.createObjectStore("notes", { keyPath: "id", autoIncrement:
  true });
➤ };

  request.onsuccess = function(event) {
    db = event.target.result;
    fetchNotes();
  };
  request.onerror = function(event){
    alert(event.debug[1].message);
  }
};
```

Метод createObjectStore() определяет используемую таблицу базы данных. При вызове передается имя таблицы, за которым следует ассоциативный массив параметров. В нашем примере параметры указывают, что каждая запись должна иметь уникальный, автоматически увеличиваемый ключ с именем id. Команда SQL выполняется в транзакции с двумя методами обратного вызова: для успешного выполнения и для ошибки. Эта схема будет использоваться для всех действий.

В будущем при изменении значения version снова будет активизирован метод обратного вызова onupgradeneeded(). Это позволяет организовать простое обновление схем на машине пользователя.

Итак, таблица готова, и теперь приложение может сделать что-то полезное.

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. Наш метод `connectToDB()` обеспечивает подключение и создание базы данных, а при успешном подключении он вызывает метод `fetchNotes()`, который мы сейчас напишем.

Метод `fetchNotes()` загружает все заметки из базы данных. Он определяется следующим образом:

html5_indexedDB/javascripts/notes.js

```
var fetchNotes = function(){
  var keyRange, request, result, store, transaction;

  transaction = db.transaction(["notes"], "readwrite");
  store = transaction.objectStore("notes");

  // Получить все данные из хранилища
  keyRange = IDBKeyRange.lowerBound(0);
  request = store.openCursor(keyRange);

  request.onsuccess = function(event) {
    result = event.target.result;
    if(result){
      addToNotesList(result.key, result.value);
      result.continue();
    }
  };

  request.onerror = function(event) {
    alert("Unable to fetch records.");
  };
};
```

Метод загружает результаты из базы данных с использованием *курсора*. При обнаружении очередного результата вызывается метод `addNoteToList()`, который добавляет заметку на боковую панель. Процесс повторяется для каждой записи в базе данных благодаря вызову `result.continue()`, который получает следующую запись, что приводит к повторному вызову `onsuccess()`.

Нам нужно знать как ключ (`key`), так и данные (`data`) результата. В качестве ключа используется поле `id`, а в качестве данных — объект JavaScript, содержащий название и тело заметки. Оба значения передаются методу `addNoteToList()` для добавления заметки на боковую панель. Метод `addNoteToList()` определяется так:

html5_indexedDB/javascripts/notes.js

```
var addToNotesList = function(key, data){
  var item = $("<li>");
  var notes = $("#notes");

  item.attr("data-id", key);
  item.html(data.title);
  notes.append(item);
};
```

Идентификатор записи встраивается в пользовательский атрибут данных элемента. Мы используем этот идентификатор для поиска загружаемой записи при щелчке на элементе списка. Затем новый элемент списка загружается в неупорядоченный список с идентификатором `notes`.

Для определения пользовательского атрибута данных, в котором хранится идентификатор, используется метод jQuery `attr()`. Метод jQuery `data()` может взаимодействовать с пользовательскими атрибутами данных[1], но у метода `data()` имеются побочные эффекты. Для простоты мы обойдемся без метода jQuery `data()` в нашем приложении.

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`.

Добавьте в файл *javascripts/notes.js* следующий обработчик события:

html5_indexedDB/javascripts/notes.js

```
$("#notes").click(function(event){
  var element = $(event.target);
  if (element.is('li')) {
    getNote(element.attr("data-id"));
  }
});
```

[1] http://api.jquery .com/data/

В нем вызывается метод `getNote()`, который получает идентификатор и одну заметку из базы данных. Определение `getNote()` выглядит так:

html5_indexedDB/javascripts/notes.js

```
var getNote = function(id){
  var request, store, transaction;
  id = parseInt(id);

  transaction = db.transaction(["notes"]);
  store = transaction.objectStore("notes");

  request = store.get(id);

  request.onsuccess = function(event) {
    showNote(request.result);
  };

  request.onerror = function(error){
    alert("Unable to fetch record " + id);
  };
}
```

Этот метод имеет много общего с приведенным ранее методом `fetchNotes()`. Мы получаем объект `request`, вызывая `get()` для `store` с передачей идентификатора. Если запрос выполнен успешно, заметка выводится на форме вызовом метода `showNote()`, который определяется следующим образом:

html5_indexedDB/javascripts/notes.js

```
var showNote = function(data){
  var note = $("#note");
  var title = $("#title");

  title.val(data.title);
  title.attr("data-id", data.id);
  note.val(data.note);
  $("#delete_button").show();
}
```

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.

Создание, обновление и удаление записей

Наша система читает записи и выводит их, но нужно предусмотреть возможность ввода заметок в систему. Начнем с кнопки New, которая очищает форму, чтобы пользователь мог создать новую заметку после редактирования существующей. Сначала добавляется обработчик события щелчка на кнопке New:

html5_indexedDB/javascripts/notes.js

```
$("#new_button").click(function(event){
  newNote();
});
```

Обработчик сбрасывает атрибут `data-id` поля `title` и удаляет значения с формы. Также в интерфейсе формы скрывается кнопка `Delete`.

html5_indexedDB/javascripts/notes.js

```
var newNote = function(){
  var note = $("#note");
  var title = $("#title");

  $("#delete_button").hide();
  title.removeAttr("data-id");
  title.val("");
  note.val("");
}
```

Когда пользователь щелкает на кнопке `Save`, должен срабатывать код вставки новой или обновления существующей записи. Мы используем тот же прием, что и в предыдущем случае, — начнем с обработчика события кнопки Save:

html5_indexedDB/javascripts/notes.js

```
$("#save_button").click(function(event){
  var id, note, title;

  event.preventDefault();
  note = $("#note");
```

```
  title = $("#title");
  id = title.attr("data-id");

  if(id){
    updateNote(id, title.val(), note.val());
  }else{
    insertNote(title.val(), note.val());
  }
});
```

Метод проверяет атрибут `data-id` поля `title` формы. Если идентификатор отсутствует, форма считает, что мы вставляем новую запись, и вызывает метод `insertNote()`.

html5_indexedDB/javascripts/notes.js

```
var insertNote = function(title, note){
  var data, key

  data = {
    "title": title,
    "note": note,
  };

  var transaction = db.transaction(["notes"], "readwrite");
  var store = transaction.objectStore("notes");
  var request = store.put(data);

  request.onsuccess = function(event) {
    key = request.result;
    addToNotesList(key, data);
    newNote();
  };
};
```

Метод `insertNote()` вставляет запись в базу данных и использует свойство `key` результата для получения идентификатора только что созданной записи. Напомним, что поле было определено как автоматически увеличиваемое, поэтому при создании новой записи база данных предоставляет для этой записи новый уникальный идентификатор. Далее вызов метода `addToNotesList()` включает заметку в список на боковой панели, с передачей ключа и остальных данных. Также он вызывает метод `newForm()` для очистки формы.

На следующем шаге выполняется обработка обновлений. Метод `updateNote()` очень похож на остальные методы, добавленные ранее.

html5_indexedDB/javascripts/notes.js

```
var updateNote = function(id, title, note){
  var data, request, store, transaction;
  id = parseInt(id);
  data = {
    "title": title,
    "note": note,
    "id" : id
  };

  transaction = db.transaction(["notes"], "readwrite");
  store = transaction.objectStore("notes");
  request = store.put(data);

  request.onsuccess = function(event) {
    $("#notes>li[data-id=" + id + "]").html(title);
  };
};
```

Если команда обновления была выполнена успешно, мы обновляем заголовок заметки в списке; для этого используем jQuery для поиска элемента боковой панели, у которого атрибут `data-id` совпадает со значением только что обновленного идентификатора.

Удаление записей происходит практически так же. Нам понадобится обработчик для события `delete`.

html5_indexedDB/javascripts/notes.js

```
$("#delete_button").click(function(event){
  var title = $("#title");
  event.preventDefault();
  deleteNote(title.attr("data-id"));
});
```

Сам метод `delete()` не только удаляет запись из базы данных, но и исключает ее из списка заметок на боковой панели.

html5_indexedDB/javascripts/notes.js

```
var deleteNote = function(id){
  var request, store, transaction;
```

```
  id = parseInt(id);

  transaction = db.transaction(["notes"], "readwrite");
  store = transaction.objectStore("notes");
  request = store.delete(id);

  request.onsuccess = function(event) {
    $("#notes>li[data-id=" + id + "]").remove();
    newNote();
  };
};
```

Метод также вызывает `newForm()` для очистки формы, чтобы создание новой записи не привело к случайному дублированию существующей.

Наконец, необходимо реализовать удаление всех заметок. Мы добавляем обработчик события для щелчка на кнопке Delete All:

html5_indexedDB/javascripts/notes.js

```
$("#delete_all_button").click(function(event){
  clearNotes();
});
```

В этом обработчике вызывается метод `clearNotes()`, удаляющий все записи из базы данных. Метод `clearNotes()` строится по уже знакомой схеме:

html5_indexedDB/javascripts/notes.js

```
var clearNotes = function(id){
  var request, store, transaction;

  transaction = db.transaction(["notes"], "readwrite");
  store = transaction.objectStore("notes");
  request = store.clear();
  request.onsuccess = function(event) {
    $("#notes").empty();
  };

  request.onerror = function(event){
    alert("Unable to clear things out.");
  }
};
```

С этим последним изменением наше приложение готово. Чтобы весь механизм заработал, следует вызвать метод `connectToDB()` и загрузить записи из базы данных. Также необходимо вызвать метод `newForm()`,

чтобы подготовить форму к использованию. При этом также скрывается кнопка Delete.

html5_indexedDB/javascripts/notes.js

```
connectToDB();
newNote();
```

Вот и все! Впрочем, с этим приложением необходима осторожность. Так как данные хранятся на стороне клиента, они могут быть легко удалены при очистке клиентом кэша и локальных хранилищ данных (как и в случае с cookie). Кроме того, хранилище данных не следует за пользователем с компьютера на компьютер. Для решения этих проблем необходимо реализовать механизм синхронизации данных с сервером, а эта тема выходит за рамки книги.

Наше приложение работает в Internet Explorer 10 и настольных системах с Chrome и Firefox. Однако Safari, мобильные браузеры Android и Safari для iOS не поддерживают IndexedDB. В них поддерживается устаревший (но широко используемый) стандарт Web SQL Databases. Internet Explorer 8 и 9 не поддерживают ни Web SQL Databases, ни IndexedDB. Давайте посмотрим, как действовать в такой ситуации

Обходное решение

Самый простой выход — уговорить пользователей таких приложений заменить свои браузеры чем-то, что поддерживает IndexedDB — например, Internet Explorer, а если это невозможно из-за ограничений операционной системы — последней версией Chrome. В такой практике нет ничего необычного, особенно когда переход на альтернативный браузер позволяет построить внутреннее приложение, которое будет работать также и на мобильных устройствах.

Конечно, такие организационные меры доступны не всегда. Но мы можем использовать IndexedDBShim, чтобы наше приложение заработало в браузерах с поддержкой спецификации Web SQL Databases[1]. Это означает, что приложение будет работать на мобильных устройствах (iPhone, iPad, планшеты Android), потому что эти браузеры поддерживают Web SQL Databases. Чтобы использовать IndexedDBShim, загрузите компактную версию и включите ее в секцию `<head>` страницы:

[1] http://nparashuram.com/IndexedDBShim/

html5_indexedDB/index.html

```
<script src="javascripts/IndexedDBShim.min.js"></script>
```

Тег должен размещаться в секции <head> страницы, в противном случае в Safari его функциональность может оказаться недоступной в самом начале работы с самыми непредсказуемыми сбоями. Вы можете использовать Modernizr для загрузки этой библиотеки только в случае необходимости, но тогда возникает вопрос: насколько разумно загружать библиотеку только для того, чтобы загрузить другую библиотеку, которая, к тому же, выполняет собственную проверку? Если вы используете библиотеку Modernizr для других целей, то ее применение для загрузки всех библиотек по мере надобности вполне оправдано.

С таким обходным решением приложение будет работать в Safari или других браузерах, поддерживающих Web SQL. Впрочем, решение не идеально, поэтому его стоит тщательно протестировать, но это работоспособное и распространенное решение для браузеров, поддерживающих Web SQL.

К сожалению, обходного решения, поддерживающего функциональность IndexedDB для пользователей Internet Explorer 8 и 9, не существует. Пользователям этих браузеров можно только посочувствовать.

Наряду с сохранением данных в браузере клиента также можно создавать приложения, вообще не требующие активного подключения к Интернету. Следующий рецепт посвящен автономным приложениям.

WEB SQL DATABASES

Рабочий проект спецификации Web SQL Databases определяет API, позволяющий разработчикам использовать в браузере базу данных на основе SQLite — популярной базы данных с открытым кодом, часто используемой на платформах iOS и Android. Этот программный интерфейс реализован в Safari, Safari для iOS, браузерах Android и Chrome; он чрезвычайно прост в использовании, если вы умеете писать команды SQL.

К сожалению, поскольку спецификация Web SQL Databases ориентирована на одну конкретную реализацию, комитет по стандартизации решил приостановить ее развитие до тех пор, пока не появится конкурирующая реализация. Mozilla и Microsoft решили вместо этого продвигать IndexedDB, а компания Google добавила IndexedDB в последние версии Chrome.

Но если вас интересует, как работает технология Web SQL Databases, в папке html5_websql архива исходного кода книги находится соответствующая реализация приложения notes. Эта технология может быть чрезвычайно полезной, если вы разрабатываете приложения на платформах с ее поддержкой — даже при том, что работа над самой спецификацией приостановлена.

Рецепт 29. 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, или просто для построения приложений, которые должны работать в автономном режиме.

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

Наше приложение содержит большое количество зависимостей. При загрузке главной страницы HTML браузер должен загрузить файлы CSS, а также файлы с кодом JavaScript. Вы можете создать файл *манифеста* со списком 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.

Давайте построим манифест для нашего приложения, чтобы его можно было использовать в автономном режиме. Создайте файл с именем *notes.appcache*. Его содержимое должно выглядеть так.

html5_offline/notes.appcache

```
CACHE MANIFEST
# v = 1.0.0
stylesheets/style.css
javascripts/notes.js
javascripts/IndexedDBShim.min.js
javascripts/jquery-1.9.1.min.js
```

[1] http://www.w3.org/TR/html5/browsers.html#offline

Изменение комментария с номером версии сообщает браузеру о необходимости загрузки новых версий файлов. Обновления временно́й метки недостаточно; необходимо изменить фактическое содержимое файла. Для этого и нужен комментарий `version`.

Чтобы приложение работало автономно, вариант с размещением jQuery в Google уже не подходит. Следовательно, мы должны загрузить jQuery и изменить тег `<script>`, чтобы библиотека jQuery загружалась из папки *javascripts*.

html5_offline/index.html

```
<script src="javascripts/jquery-1.9.1.min.js"></script>
<script src="javascripts/notes.js"></script>
```

Файл манифеста необходимо связать с документом HTML. Для этого элемент `html` приводится к следующему виду:

html5_offline/index.html

```
<html manifest="notes.appcache">
```

Вот и все! Включив консоль в Chrome, вы увидите, что манифест работает (рис. 32).

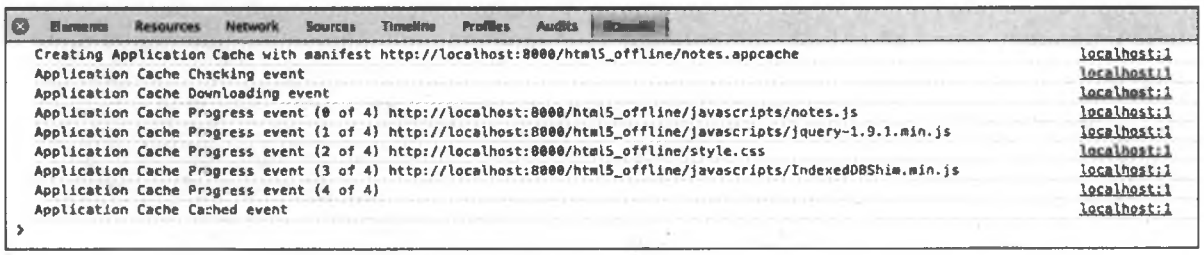


Рис. 32. Кэшированные файлы на консоли

После кэширования всех файлов пользователь может отключиться от Интернета — приложение продолжит работать. Впрочем, есть одна загвоздка — файл манифеста должен предоставляться веб-сервером, потому что манифест должен поставляться с типом MIME `text/cache-manifest`. Сервер Node.js в примерах исходного кода решает эту проблему при тестировании, но если вы используете Apache, укажите тип MIME в файле *.htaccess*.

html5_offline/.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 может 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*:

html5_offline/.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.

Проверка сетевого подключения

Написанное нами приложение работает автономно, но что если вы захотите обнаружить активное сетевое подключение и синхронизировать локальные данные с сервером?

Сначала при помощи свойства `navigator.onLine` можно определить, доступно ли подключение в настоящее время.

html5_offline/offlinetest.html

```
if (navigator.onLine) {
  alert('online')
} else {
  alert('offline');
}
```

Такое решение работает в Safari, Firefox и Chrome; оно просто в использовании, но это всего лишь простая проверка сетевого подключения. С ним можно узнать текущий статус подключения. Прослушивание событий также позволяет организовать обработку разрыва связи:

html5_offline/offlinetest.html

```
window.addEventListener("offline", function(e) {
  alert("offline");
}, false);
window.addEventListener("online", function(e) {
  alert("online");
}, false);
```

Этот прием позволяет обнаружить потерю связи, вывести сообщение, а затем синхронизировать данные, когда связь восстановится.

Так как синхронизация данных в приложении потребует программирования на уровне исполнительной подсистемы, мы не будем рассматривать эту тему. Впрочем, у вас имеется все необходимое для дальнейших исследований.

Перспективы

Такие технологии, как Web Storage и IndexedDB, 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.

10 Взаимодействие с другими API

Наряду с новой разметкой, стилями и мультимедийными возможностями в HTML5 и CSS3 определены чрезвычайно мощные программные интерфейсы для создания более функциональных, более мощных веб-приложений. Тема хранения данных на клиентской машине уже была затронута в предыдущей главе, а сейчас мы пойдем еще дальше. Мы 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. Глава завершается описанием поддержки перетаскивания (Drag and Drop API) в HTML5.

Многие интересные API, существование которых началось со спецификации HTML5, со временем преобразовались в отдельные проекты. Другие настолько тесно связаны с HTML5, что разработчикам становится трудно различить их. Объединяя то, что вы уже знаете, с новыми возможностями, вы сможете сделать работу пользователя более простой и удобной.

В этой главе рассматриваются следующие API:

History

Управление историей просмотра. [C5, F3, S4, IE8, O10.1 IOS3.2, A2]

Cross-Document Messaging

Передача сообщений между окнами и тегами `<iframe>` с контентом, загруженным в разных доменах. [C5, F4, S5, IOS4.1, A2]

Web Sockets
: Создание подключения с поддержкой состояния между браузером и сервером. [C5, F6, S5, IE10, O12.1, IOS6]

Geolocation
: Получение информации о широте и долготе. [C5, F3.5, S5, O10.6, IOS3.2, A2.1]

Drag-and-Drop
: Поддержка перетаскивания. [C4, F3.5, S3.1, IE6 (частичная), IE10 (полная), O12]

Рецепт 30. История просмотра

В спецификации HTML5 представлен API для управления историей просмотра[1]. С его помощью можно добавлять новые записи в историю, заменять их и даже сохранять данные, которые могут загружаться при повторном посещении страницы. Данная возможность прекрасно подходит для одностраничных приложений, которые обновляются динамически, но при этом должны поддерживать функциональность кнопки Back.

В рецепте 15 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. Если щелкнуть на вкладке, то нажатие кнопки Back в браузере вернет вас не к предыдущей вкладке, а к ранее посещенной странице. History API позволяет решить эту проблему.

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. Текущий прототип домашней страницы уже содержит код переключения вкладок; нам остается лишь добавить код сохранения вкладки, выбранной пользователем. Создайте новый метод с именем `addTabToHistory()`:

html5_history/javascripts/application.js

```
var addTabToHistory = function(target){
  var tab = target.attr("href");
  var stateObject = {tab: tab};
  window.history.pushState(stateObject, "", tab);
}
```

Функция получает вкладку, на которой щелкнул пользователь. Значение атрибута `href` используется для сохранения состояния просмотра методом `pushState()`, получающим три аргумента. В первом параметре 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 (свойство `tab`) сохраняется запись `"#services"`.

[1] http://www.w3.org/TR/html5/browsers.html#history

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. Большинство браузеров не использует эту информацию, так что мы просто передаем пустую строку.

В третьем аргументе передается URL-адрес, который должен отображаться в строке заголовка. Мы снова используем идентификатор вкладки, потому что в URL-адрес будет добавлен знак #. Если бы мы работали с сервером исполнительной подсистемы и использовали Ajax, возможно, лучше было бы передать относительный URL-адрес вида `/about` или `/services`. В этом случае при прямом переходе к URL-адресу исполнительная подсистема смогла бы отреагировать соответствующим образом. Так как в нашем приложении такие статические страницы не используются, мы не можем просто повозиться с URL-адресом и заставить его работать.

Чтобы активизировать новый код, мы включаем метод `addTabToHistory()` в обработчик щелчка на ссылках. При вызове ему передается вкладка, на которой щелкнул пользователь:

html5_history/javascripts/application.js

```
$("nav ul").click(function(event){
  var target = $(event.target);
  if(target.is("a")){
    event.preventDefault();
    if ( $(target.attr("href")).attr("aria-hidden")){

addTabToHistory(target);

      activateTab(target.attr("href"));
    };
  };
});
```

Добавить новое состояние в историю просмотра недостаточно — нужно еще написать код, выполняемый при нажатии кнопки Back.

Чтение предыдущего состояния

Когда пользователь щелкает на кнопке Back, срабатывает событие `window.onpopstate()`. Мы используем этот обработчик для отображения вкладки, сохраненной в объекте состояния.

html5_history/javascripts/application.js

```
var configurePopState = function(){
  window.onpopstate = function(event) {
    if(event.state){
      var tab = (event.state["tab"]);
      activateTab(tab);
    }
  };
};
```

Все, что от нас требуется — извлечь информацию о вкладке из объекта, хранимого в истории просмотра, и передать ее функции `activateTab()`. Да здравствует повторное использование кода!

История по умолчанию

В этом решении необходимо исправить пару недостатков. При первой загрузке страницы состояние истории просмотра будет пустым, поэтому нам придется задать ее вручную. Кроме того, URL-адрес изменяется при щелчке на вкладке, но при нажатии кнопки Reload вместо нужной вкладки отображается вкладка Welcome. Следовательно, при загрузке страницы нужно проверить URL-адрес, посмотреть, какая вкладка открывается, и выбрать ее.

html5_history/javascripts/application.js

```
var activateDefaultTab = function(){
  tab = window.location.hash || "#welcome";
  activateTab(tab);
  window.history.replaceState( {tab: tab}, "", tab);
};
```

Если значение `location.hash` отсутствует, мы присваиваем значение по умолчанию. Затем метод `history.replaceState()` используется для выбора вкладки. Он работает аналогично `pushState()`, но вместо добавления нового элемента заменяется текущий элемент.

Чтобы улучшить структуру кода, мы создадим функцию `init()`, которая вызывает новые методы и исходный метод `configureTabSelection()`:

html5_history/javascripts/application.js

```
var init = function(){

  configureTabSelection();

configurePopState();
activateDefaultTab();
};
```

Остается вызвать `init()`, чтобы все заработало:

html5_history/javascripts/application.js

```
init();
```

Теперь при открытии страницы мы сможем перебирать открывавшиеся ранее вкладки, используя историю просмотра.

Обходное решение

Наше решение работает в Chrome, Firefox, Safari, Internet Explorer 9 и выше. Для старых браузеров лучше всего использовать библиотеку History.js[1], которая создает межбраузерную прослойку для работы этой функциональности. Однако реализация данного решения не сводится к простому подключению библиотеки. Вам придется подключить библиотеку и изменить свой код для использования ее кода вместо объекта `history` из браузера. К счастью, библиотека очень близка к спецификации и избавляет от необходимости проверять поддержку истории. Но поскольку реализация требует внесения изменений в код, мы ее рассматривать не будем.

Впрочем, нам все равно стоит предотвратить появление ошибок в старых браузерах, поэтому мы используем Modernizr для обнаружения поддержки истории. Включите Modernizr в страницу HTML:

html5_history/fallback/index.html

```
<script src="javascripts/modernizr.js"></script>
```

Затем мы используем объект `Modernizr.history` для упаковки вызовов методов объекта `window.history`.

html5_history/fallback/javascripts/application.js

```
$("nav ul").click(function(event){
  var target = $(event.target);
  if(target.is("a")){
    event.preventDefault();
    if ( $(target.attr("href")).attr("aria-hidden")){
      if(Modernizr.history){
        addTabToHistory(target);
      }
      activateTab(target.attr("href"));
    };
  };
```

[1] https://github.com/browserstate/history.js/

```
  });
var init = function(){
  configureTabSelection();
  if(Modernizr.history){
    configurePopState();
    activateDefaultTab();
  }
};
```

Если вы решите реализовать History.js, можно использовать `Modernizr.load()` для загрузки разных версий кода в зависимости от того, нужно ли вам обходное решение.

Теперь мы рассмотрим возможности обмена информацией между двумя сайтами в разных доменах.

Рецепт 31. Передача информации между доменами

Клиентским веб-приложениям всегда запрещалось прямое взаимодействие со сценариями других доменов. Это ограничение было установлено ради безопасности пользователей[1], однако наряду с повышением безопасности оно усложняет нашу работу, потому что существует немало законных поводов для взаимодействия двух сайтов. Существует много хитроумных обходных путей, включая использование посредников на стороне сервера и манипуляции с URL. Однако теперь появился более удобный способ.

В спецификации HTML5 представлен API *Cross-Document Messaging*, или *Web Messaging*[2], позволяющий организовать обмен 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, как на рис. 33.

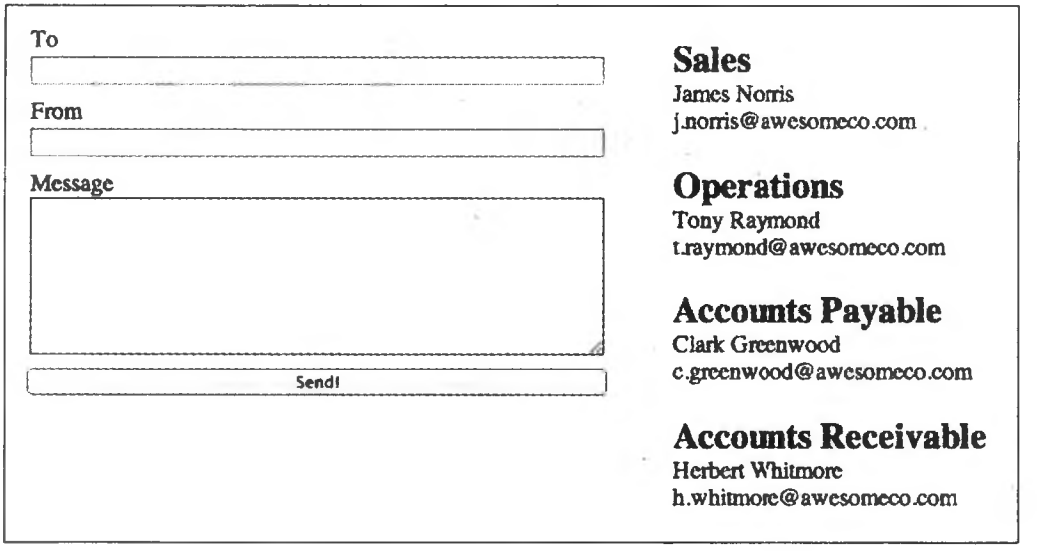


Рис. 33. Сайт поддержки

[1] Политика единого происхождения (Same Origin Policy) более подробно объясняется в статье https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript.

[2] http://www.w3.org/TR/webmessaging/#web-messaging

Эта информация будет поступать из системы управления контентом (CMS) на другом сервере, но в этом прототипе список контактов будет встроен 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 два веб-сервера. Примеры, которые мы будем рассматривать, не работают в некоторых браузерах без поставки страниц с веб-сервера. Чтобы вы могли убедиться в том, что междокументный обмен сообщениями действительно работает, приложение со списком контактов и сайт поддержки как минимум должны работать на разных портах. В архив исходного кода книги включен простой сценарий, который может использоваться для запуска веб-серверов для данного рецепта. Для этого потребуется библиотека Node.js (о том, как настроить и запустить этот сервер, рассказано на с. 16). В коде нашего примера будут использоваться следующие URL-адреса и порты этого сервера:

- Список контактов будет поставляться с `localhost:4000`.
- Страница поддержки будет поставляться с `localhost:3000`.

Если вы собираетесь разместить файлы на отдельных серверах, находящихся под вашим управлением, просто измените URL в коде.

Список контактов

Начнем с создания списка контактов. Мы используем файл, встроенный в тег `<iframe>` сайта поддержки. Базовая разметка выглядит так.

html5_cross_document/contactlist/index.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>Contact List</title>
    <link rel="stylesheet" href="stylesheets/style.css" >
  </head>
  <body>
    <ul id="contacts">
      <li>
        <h2>Sales</h2>
        <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>
  </body>
</html>
```

Страница также загружает библиотеку jQuery и файл *application.js* непосредственно перед закрывающим тегом <body>.

html5_cross_document/contactlist/index.html

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

<script src="javascripts/application.js"></script>
```

Таблица стилей для списка контактов выглядит так.

html5_cross_document/contactlist/stylesheets/style.css

```
ul{
list-style: none;
}

ul h2, ul p{margin: 0;}

ul > li{margin-bottom: 20px;}
```

С этой парой настроек 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. Напомню, что список контактов поставляется с *http://localhost:4000*, но включается в `<iframe>` страницей, поставляемой с *http://localhost:3000*. Это и есть URL-адрес, по которому будут отправляться сообщения.

Весь обработчик событий выглядит так.

html5_cross_document/contactlist/javascripts/application.js

```
$("#contacts li").click(function(event){
  var email, origin;
  email = $(this).find(".email").html();
  origin = "http://localhost:3000/index.html";
  if(window.postMessage){
    window.parent.postMessage(email, origin);
  }
});
```

Если вы воспроизводите код примеров на своем компьютере, не забудьте изменить переменную `origin` — по причинам, связанным с безопасностью, в ней должен содержаться URL-адрес родительского окна.

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 контактов и `<iframe>` со списком контактов.

html5_cross_document/supportpage/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://localhost:4000/index.html"></iframe>
</div>
```

Стилевое оформление реализуется следующим кодом CSS, который мы включим в файл *o stylesheets/style.css*:

html5_cross_document/supportpage/stylesheets/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 к полям формы применяются базовые стили.

Получение сообщений

Событие `onmessage` инициируется при получении сообщения текущим окном. Сообщение передается в свойстве объекта `event`. Для регистрации события будет использоваться метод jQuery `on()`, чтобы решение одинаково работало во всех браузерах.

html5_cross_document/supportpage/javascripts/application.js

```
$(window).on("message",function(event){
  $("#to").val(event.originalEvent.data);
});
```

Метод jQuery `on()` инкапсулирует событие и блокирует доступ к некоторым из его свойств. Нужную информацию можно получить из свойства `originalEvent` объекта события.

Если вы откроете сайт поддержки в Firefox, Chrome, Safari или Internet Explorer 8 и выше, то увидите, что прекрасно работает. В обходном решении нет необходимости. Если у вас возникнут проблемы, запустите сценарий веб-сервера из примеров кода и откройте адрес *http://localhost:3000*.

Взаимодействие между двумя страницами или приложениями открывает массу возможностей для создания более модульных приложений.

Обходное решение

Хотя Internet Explorer версий 8 и 9 поддерживает Cross-Document Messaging, функция `postMessage()` может работать только со строками. Она не поддерживает работу с объектами. Кроме того, сообщения могут передаваться только между `frame` и `iframe`. Возможно, вам пригодится плагин jQuery PostMessage, так как он поддерживает сериализацию объектов[1].

Итак, теперь вы знаете, как отправлять сообщения между сайтами. Давайте посмотрим, как организовать двустороннее общение пользователей.

[1] http://benalman.com/projects/jquery-postmessage-plugin/

Рецепт 32. 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). Окно чата показано на рис. 34.

Рис. 34. Интерфейс чата

[1] Технология Web Sockets была выделена в отдельную спецификацию, которая находится по адресу http://www.w3.org/TR/websockets/.

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. Веб-сервер из примеров кода запускает чат-сервер, который мы сможем использовать для тестирования.

Интерфейс чата

Мы создадим очень простой интерфейс, показанный на рис. 34, 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.

html5_websockets/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My Chat Server</title>
    <link rel="stylesheet" href="stylesheets/style.css">
  </head>
  <body>
    <div id="chat_wrapper">
      <h2>AwesomeCo Help!</h2>
      <form id="nick_form" action="#" method="post">
        <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">
        <p>
          <label>Message
            <input id="message" type="text" />
```

```
          </label>
          <input type="submit" value="Send">
        </p>
      </form>
    </div>
  </body>
</html>
```

Также необходимо добавить ссылки на jQuery и файл JavaScript, содержащий код взаимодействия с сервером Web Sockets. Эти ссылки должны размещаться непосредственно над закрывающим тегом `<body>`.

html5_websockets/index.html

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

Таблица стилей, созданная в *stylesheets/style.css*, содержит следующие определения:

html5_websockets/stylesheets/style.css

```
1  #chat_wrapper{
-    background-color: #ddd;
-    height: 440px;
-    padding: 10px;
-    width: 320px;
-  }
-
-  #chat_wrapper h2{ margin: 0; }
-
10 #chat{
-    background-color: #fff;
-    height: 300px;
-    overflow: auto;
-    padding: 10px;
-    width: 300px;
-  }
```

В строке 13 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.

Событие	Описание
onopen()	Успешное создание подключения к серверу
onmessage()	Отправка сообщения через подключение к серверу
onclose()	Потеря или закрытие подключения к серверу

В файле *javascripts/chat.js* мы сначала создаем подключение к серверу Web Sockets.

html5_websockets/javascripts/chat.js

```
var setupChat = function(){
  // Замените фактическим IP-адресом сервера
  var webSocket = new WebSocket('ws://192.168.1.2:9394/');
};
```

Все эти обработчики событий помещаются в функцию `setupChat()`. Это улучшает структуру кода и позволяет нам управлять моментом времени, когда будут присоединяться события. Позднее нам понадобится проверить поддержку сокетов браузером и загрузить обходное решение, так что эти события не должны срабатывать до того момента, когда мы будем к этому готовы.

Если подключение к серверу создано успешно, необходимо сообщить об этом пользователю. Метод `onopen()` определяется следующим образом внутри функции `setupChat()`:

html5_websockets/javascripts/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()` (также находящееся внутри функции `setupChat()`):

html5_websockets/javascripts/chat.js

```
1  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. В строке 3 jQuery используется для прокрутки окна, чтобы новое сообщение выводилось внизу.

Далее необходимо обеспечить обработку разрыва связи. Метод `onclose()` срабатывает при закрытии подключения.

html5_websockets/javascripts/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. Этот обработчик тоже размещается в функции `setupChat()`:

html5_websockets/javascripts/chat.js

```
$("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/javascripts/chat.js

```
$("form#nick_form").submit(function(e){
  e.preventDefault();
  var textfield = $("#nickname");
  webSocket.send("/nick " + textfield.val());
});
```

Наконец, необходимо вызвать функцию `setupChat()` для вызова всех этих событий.

html5_websockets/javascripts/chat.js

```
setupChat();
```

В более сложном приложении метод `setupChat()` следовало бы разбить на отдельные методы, каждый из которых выполняет свою операцию. Но в данном случае такая структура работает и демонстрирует концепцию, которую мы изучаем.

Вот и все! Пользователи Safari, Chrome и Firefox 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 библиотеки web-socket-js[2] и поместите ее в проект. Мы снова используем Modernizr для проверки функциональности и загрузки файлов JavaScript на нашей странице. Сначала Modernizr добавляется на страницу с использованием адаптированной версии с функцией `load()`, которую мы использовали в главе 3:

[1] http://github.com/gimite/web-socket-js/

[2] https://github.com/gimite/web-socket-js/archive/master.zip

html5_websockets/index.html

```
<script src='javascripts/modernizr.js'></script>
```

Затем в файле *javascripts/chat.js* метод `Modernizr.load()` используется для загрузки и настройки обходного решения:

html5_websockets/javascripts/chat.js

```
Modernizr.load(
{
test: Modernizr.websockets,
nope:
{
"swfobject" : "web-socket-js/swfobject.js",
"websocket" : "web-socket-js/web_socket.js"
},
callback: function(url, result, key){
if (!result){
if(key === "swfobject"){
WEB_SOCKET_SWF_LOCATION = "web-socket-js/WebSocketMain.swf";
WEB_SOCKET_DEBUG = true;
}
}
},
complete: function(){

      setupChat();

}
}
);
```

Если браузер не поддерживает сокеты, то загружается библиотека, но нам также необходимо задать значение переменной, определяющей местонахождение файла *WebSocketMain*; мы делаем это в `callback()`, но только в том случае, если поддержка сокетов недоступна.

Здесь загружаются два разных сценария, и `callback()` срабатывает по одному разу для каждого загруженного сценария. Modernizr позволяет пометить каждый сценарий ключом, который передается методу обратного вызова. По ключу можно определить, какой сценарий был загружен. В нашем примере ключ используется для идентификации загрузки сценария web-socket, чтобы мы могли задать переменные, необходимые для обходного решения.

Независимо от того, поддерживает ли браузер сокеты, метод `setupChat()` всегда должен выполняться для запуска чата. Для этого метод `setupChat()` перемещается в метод обратного вызова `complete()`. Этот метод выполняется в конце процесса загрузки независимо от результата тестирования. Это идеальное место для кода, который должен выполняться всегда.

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. Отправляемый запрос содержит следующие данные:

```
<policy-file-request/>
```

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. Лучше предоставлять его через порт 843, потому что Flash Player всегда сначала отправляет запрос на этот порт.

В примерах кода этого раздела имеется простой сервер политики сокетов Flash, который может использоваться для тестирования. За информацией о настройке Node.js для тестирования в вашей среде обращайтесь к разделу «Node.js и сервер примеров» предисловия. Когда сервер заработает, вы сможете поэкспериментировать с чат-сервером по адресу *http://localhost:8000/html5_websockets/index.html*. Чтобы было еще интереснее, подключитесь к нему из Internet Explorer 8 и Chrome (рис. 35).

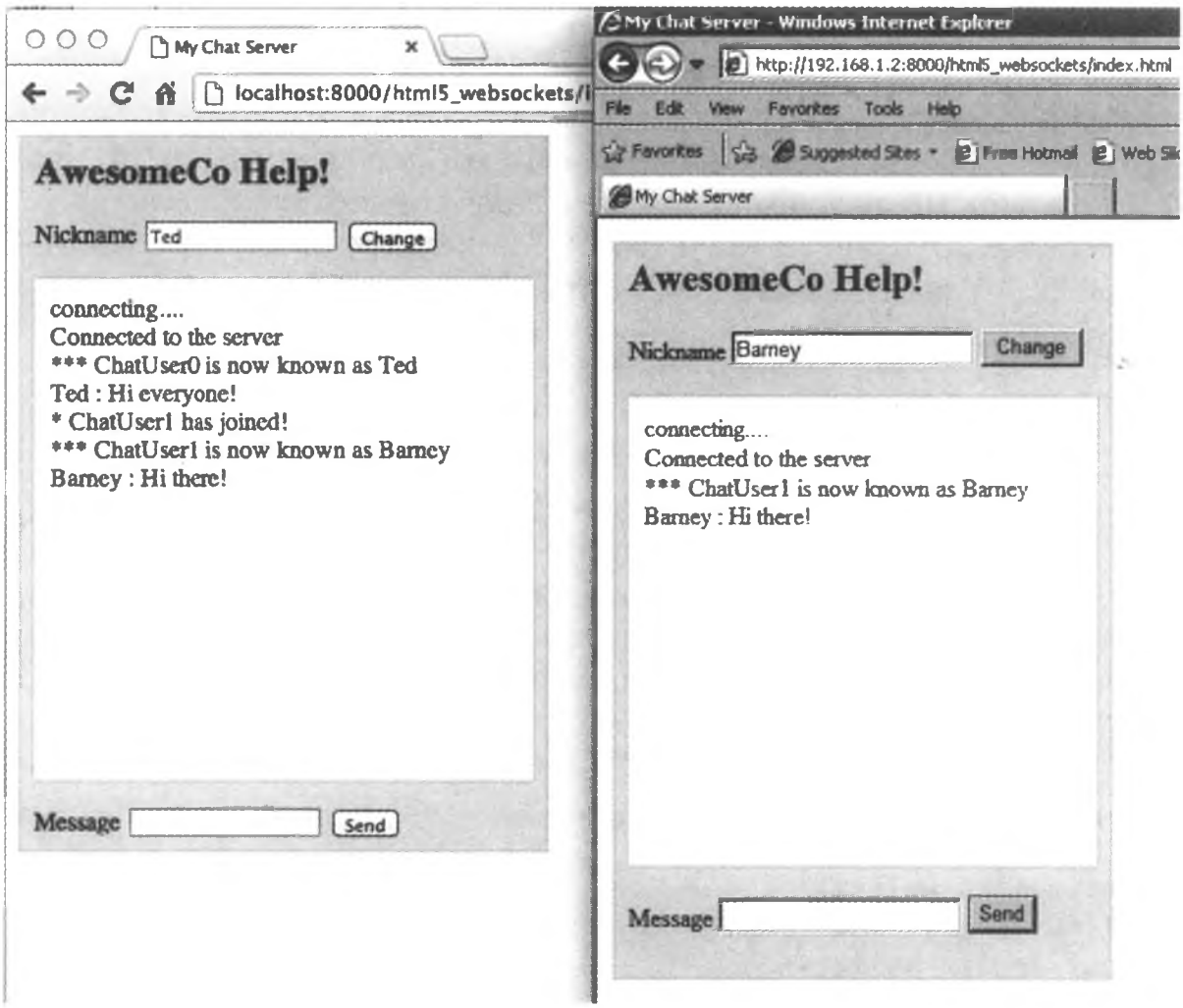


Рис. 35. Чат в разных браузерах

Впрочем, чат-сервер — это только начало. Технология Web Sockets предоставляет мощный и простой механизм передачи данных в браузеры посетителей.

Если для тестирования используются виртуальные машины, проследите за тем, чтобы использовался IP-адрес компьютера, на котором работает сервер; `localhost` работать не будет.

Мы не будем разбирать эту процедуру во всех подробностях, однако вам стоит самостоятельно изучить код чат-сервера. Он находится в файле *lib/chat.js*.

Чат-сервер — всего лишь начало. Технология Web Sockets наконец-то предоставила нам простой, но мощный способ передачи данных в браузеры пользователей в реальном времени. В следующем рецепте браузер будет использоваться для определения текущих значений широты и долготы.

```
&amp;markers=color:green|label:B|22+Southwest+3rd+Avenue,
      Portland,+OR
&amp;markers=color:green|label:C|77+West+Wacker+Drive+Chicago
      +IL">
```

Рис. 36. Текущее местонахождение отмечено маркером Y

Сначала мы определяем размер изображения, а затем сообщаем Maps API, что для получения передаваемой информации не использовалось сенсорное устройство (например, клиентские средства 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.

html5_geolocation/javascripts/geolocation.js

```
var getLatitudeAndLongitude = function(){
  navigator.geolocation.getCurrentPosition(function(position) {
    showLocation(position.coords.latitude,
                 position.coords.longitude);
  });
}
```

html5_geolocation/javascripts/geolocation.js

```
1  var getLatitudeAndLongitudeWithFallback = function(){
-    if ((typeof google === 'object') &&
-      google.loader && google.loader.ClientLocation) {
-        showLocation(google.loader.ClientLocation.latitude,
5                google.loader.ClientLocation.longitude);
-    }else{
-      var message = $("<p>Couldn't find your address.</p>");
-      message.insertAfter("#map");
-    }
10 };
```

Метод Google `ClientLocation()` в строке 3 получает данные о местоположении посетителя, а вызов метода `showLocation()` наносит координаты на карту.

Затем мы приказываем Modernizr проверить поддержку Geolocation. Если поддержка доступна, то вызывается исходный метод. Если она отсутствует, мы используем упрощенную версию `Modernizr.load()` для загрузки библиотеки Google, после чего вызываем функцию для нанесения координат 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 7). 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.

А теперь рассмотрим встроенную поддержку перетаскивания элементов в HTML5.

Рецепт 34. Перетаскивание

С момента появления компьютерной мыши пользователей учили перетаскивать элементы по экрану. Долгие годы для реализации поддержки перетаскивания в браузерах использовались решения JavaScript и DOM. В спецификации HTML5 предлагается встроенный, более эффективный (хотя и не самый простой в отношении программирования) способ реализации перетаскивания.

Руководство AwesomeCo предложило разработать программу, в которой пользователь сможет делать записи на виртуальных карточках, а затем размещать их на экране. Нам поручено разработать заготовку пользовательского интерфейса; это отличная возможность поэкспериментировать с встроенными средствами перетаскивания. Когда работа будет завершена, результат будет выглядеть примерно так, как показано на рис. 37.

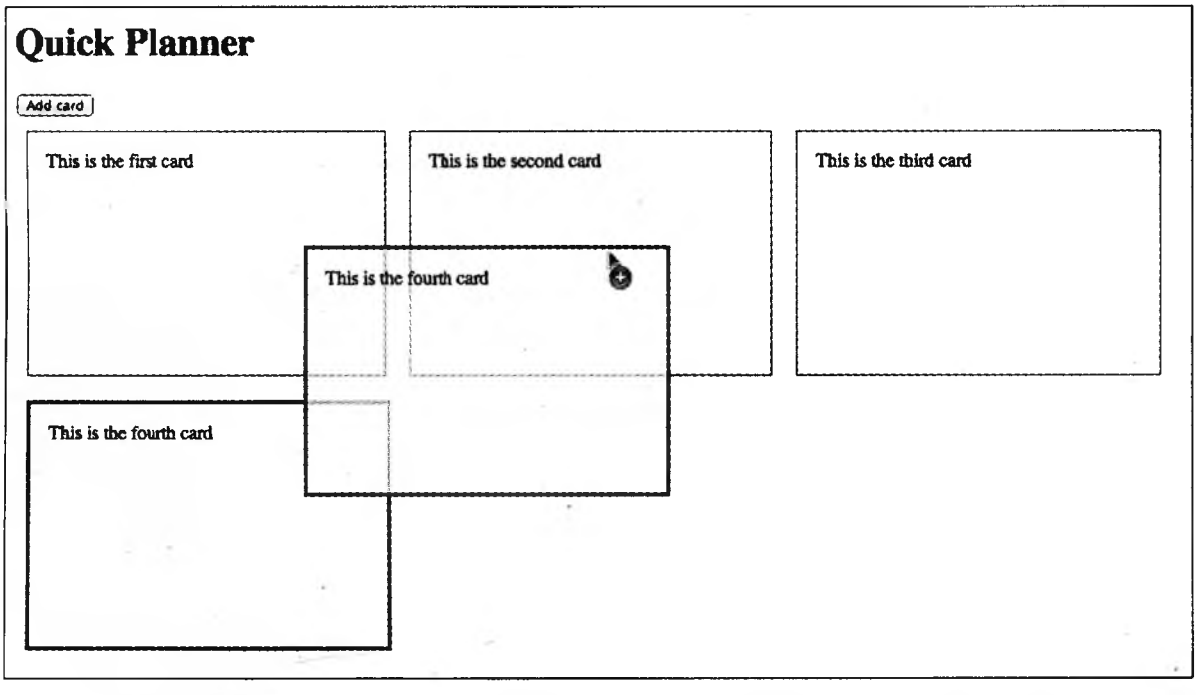


Рис. 37. Приложение с возможностью перетаскивания карточек

Создание базового интерфейса

Начнем с построения простейшей заготовки HTML, включающей ссылку на таблицу стилей *stylesheets/style.css*, кнопку для добавления карточки и область экрана, в которой будут создаваться и перемещаться карточки.

html5_dragdrop/index.html

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset="utf-8">
    <title>AwesomeCards</title>
    <link rel="stylesheet" href="stylesheets/style.css">
  </head>
  <body>
    <h1>Quick Planner</h1>
    <input type="button" id="addcard" value="Add card">

    <div id="cards">
    </div>

  </body>
</html>
```

На следующем шаге мы создадим код CSS приложения в файле stylesheets/style.css. Карточки создаются из кода JavaScript при щелчке на кнопке Add Card, при этом разметка карточки выглядит так:

```
<div class="card" draggable="true" id="card1">
  <div class="editor" contenteditable="true"></div>
</div>
```

Каждая карточка представляется элементом , вложенным в другой элемент . Во внутреннем пользователь вводит текст заметки, поэтому мы применим стилевое оформление для создания промежутка между двумя элементами.

html5_dragdrop/stylesheets/style.css

```
.card{
  background-color: #ffc;
  border: 1px solid #000;
  float: left;
  height: 200px;
  margin: 10px;
  width: 300px;
}

.editor{
  border: none;
```

```
  margin: 5%;
  width: 90%;
  height: 80%;
}

.editor:focus{ background-color: #ffe; }
.card:active{ border: 3px solid #333; }
```

Псевдокласс `:active` используется для изменения границы карточки, когда пользователь щелкает на элементе.

Добавление карточек в интерфейс

Сначала мы добавим код создания новой карточки при щелчке на кнопке Add Card. Для этого мы используем фрагмент кода jQuery, который будет упакован в отдельную функцию:

html5_dragdrop/javascripts/cards.js

```
1  addCardClickHandler = function(){
-   window.currentCardIndex = window.currentCardIndex || 0;
-   $("#addcard").click(function(event){
-     event.preventDefault();
5
-       var card = $("<div></div>")
-         .attr("id", "card" + (window.currentCardIndex++))
-         .attr("class", "card")
-         .attr("draggable", true);
10
-       var editor = $("<div></div>")
-         .attr("contenteditable", true)
-         .attr("class", "editor");
-
15      card.append(editor);
-       card.appendTo($("#cards"));
-    });
-  };
```

При добавлении карточки ей присваивается уникальный идентификатор. Можно было бы придумать какой-нибудь хитроумный механизм, но мы используем простой счетчик. В строке 2 мы ссылаемся на переменную `_currentCardIndex` объекта `window`. При первом вызове функции переменная инициализируется 0. Она объявляется в объекте `window`, чтобы значение было доступно при последующих вызовах функции. Помните,

что объект `window` имеет глобальную видимость. В более сложной ситуации мы могли бы создать собственный объект `Application` и сохранить значения в нем, чтобы избежать загрязнения глобального пространства имен.

Далее мы создаем новый элемент для карточки. Внешний элемент `div` представляет карточку, а у внутреннего элемента `div` установлен атрибут `contenteditable`. Он представляет текст карточки. Мы используем функцию jQuery для создания элемента `card`, а затем в строке 7 генерируем идентификатор, увеличивая `_currentCardIndex` на 1.

Наконец, мы применяем к элементу класс `card`, устанавливаем атрибут `draggable` и присоединяем карточку к области `cards`. После этого щелчок на кнопке `Add Card` добавляет в пользовательский интерфейс новую карточку. Благодаря установке атрибута `contenteditable` пользователь может щелкнуть на карточке и ввести ее текст.

Перемещение карточек

Прежде чем писать код, обеспечивающий перемещение карточек, мы сначала посмотрим, как будет работать этот механизм. При перетаскивании одной карточек на другую перетаскиваемая карточка вставляется после карточки, на которую она накладывается. Для этого идентификатор перетаскиваемой карточки передается нижнему элементу. При отпускании кнопки мыши jQuery находит исходный элемент по идентификатору и перемещает его в заданную позицию.

Спецификация Drag and Drop поддерживает следующие события.

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
ondragleave	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

В ходе написания кода проекта нам понадобится только пара этих событий.

Мы начнем с создания новой функции `createDragAndDropEvents()`, которая будет содержать все обработчики событий, необходимые для функциональности перетаскивания. Прежде всего мы воспользуемся jQuery для получения области страницы `cards`. В этой области будут вставляться создаваемые карточки.

html5_dragdrop/javascripts/cards.js

```
  var createDragAndDropEvents = function(){
  var cards = $("#cards");
};
```

В этом методе будут определяться наши события. Когда пользователь начинает перетаскивать карточку, срабатывает метод `ondragstart()`. Мы должны написать обработчик этого события, который будет получать идентификатор карточки и сохранять его в объекте `dataTransfer`. Этот объект содержит данные, перетаскиваемые с одного элемента на другой. Реализация будет использовать метод `setData()`.

Итак, мы должны отслеживать событие `ondragstart()` для каждой карточки. При добавлении отдельного обработчика для каждой карточки излишек обработчиков влияет на быстродействие, поэтому мы используем jQuery для создания делегированных событий. Событие создается для области `cards`, но мы приказываем jQuery делегировать его для каждой конкретной карточки. Это событие также регистрируется для новых карточек, которые добавляются пользователем позднее.

Включите в функцию `createDragAndDropEvents()` следующий код для создания обработчика события и сохранения идентификатора карточки:

html5_dragdrop/javascripts/cards.js

```
cards.on("dragstart", ".card", function(event){
  event.originalEvent.dataTransfer.setData('text', this.id);
});
```

Методу `setData()` должен передаваться как тип данных, так и сами передаваемые данные. Так как мы отправляем только идентификатор элемента, в качестве типа данных будет использоваться обычный текст. В первом аргументе `setData()` может передаваться любой тип MIME, но для максимальной совместимости со старыми браузерами мы используем ключевое слово `text`, которое поддерживается спецификацией и отображается на тип MIME `text/plain`. На момент публикации Internet Explorer 10 был единственным современным браузером, который не поддерживал типы MIME в этом параметре.

Когда карточка накладывается на другую карточку, мы хотим, чтобы перетаскиваемая карта размещалась после той карты, на которую она была сброшена. Для этой цели используется событие `ondrop()`, которое создается аналогично событию `ondragstart()`:

html5_dragdrop/javascripts/cards.js

```
cards.on("drop",".card", function(event){
  event.preventDefault();
  var id = event.originalEvent.dataTransfer.getData('text');
  var originalCard = $("#" + id);
  originalCard.insertAfter(this);
  return(false);
});
```

Из объекта `dataTransfer` извлекается идентификатор перетаскиваемого элемента, а метод `getData()` используется для определения типа данных. Мы создаем новый элемент jQuery и используем метод `insertAfter()` для его размещения за текущим элементом. Помните, что каждая карточка может перетаскиваться и является приемником перетаскивания.

Метод `getData()` доступен только в событии `drop()`. По соображениям безопасности спецификация не позволяет другим событиям перетаскивания получить доступ к хранимым данным. Событие `drop()` – единственный случай, в котором браузер может быть уверен, что пользователь не отменит событие.

Чтобы обеспечить правильность работы перетаскивания, мы должны предотвратить срабатывание события `dragover`, потому что перетаскивание по умолчанию *блокируется* браузером[1]. Следовательно, для перетаскивания карточек в функцию `createDragAndDropEvents()` необходимо добавить следующий код:

html5_dragdrop/javascripts/cards.js

```
cards.on("dragover", ".card", function(event){
  event.preventDefault();
  return(false);
});
```

С основной функциональностью проблема решена, но мы сталкиваемся с другим ограничением, обусловленным особенностями работы перетаскивания и `contenteditable`. Элементы не могут сочетать поддержку перетаскивания с возможностью редактирования «на месте». Чтобы обойти это ограничение, необходимо удалить атрибут `draggable` из элемента `card` при

[1] https://developer .mozilla.org/en-US/docs/Web/Reference/Events/dragover

получении фокуса редактором и вернуть его при потере фокуса редактором. Метод jQuery `parent()` позволяет легко получить нужный элемент.

html5_dragdrop/javascripts/cards.js

```
cards.on("focus",".editor" , function(event){
  $(this).parent().removeAttr('draggable');
});

cards.on("blur",".editor", function(event){
  $(this).parent().attr('draggable', true);
});
```

Наконец, мы выполняем две функции, добавляющих события.

html5_dragdrop/javascripts/cards.js

```
createDragAndDropEvents();
addCardClickHandler();
```

Мы можем добавлять карточки, редактировать их и перетаскивать для сортировки. Конечно, это решение работает не везде.

Обходное решение

Modernizr не поможет с обнаружением. Internet Explorer 8 поддерживает технологию Drag and Drop, но только для выделенного текста, ссылок и графики. Таким образом, хотя Modernizr правильно обнаруживает поддержку перетаскивания, в этом случае она не может считаться полноценной. По этой причине наша проверка будет построена на проверке того, поддерживает ли элемент `<div>` атрибут `draggable`. Если атрибут не поддерживается, то для сортировки будет использоваться метод jQuery UI `sortable()`.[1]

html5_dragdrop/javascripts/cards.js

```
1  if ('draggable' in document.createElement('div')) {
-    createDragAndDropEvents();
-    }else{
-      Modernizr.load(
5       {
-         load: "http://code.jquery.com/ui/1.10.3/jquery-ui.js",
-         callback: function(result, url, key){
-           $('#cards').sortable();
-       }
```

[1] http://jqueryui.com/sortable/

```
10      }
-     );
-   }
-
-   addCardClickHandler();
```

В строке 1 мы проверяем поддержку атрибута `draggable` для тегов `<div>`. Если он поддерживается, выполняется метод `createDragAndDropEvents()`, а если не поддерживается — метод `Modernizr.load()` используется для подключения jQuery UI.

После загрузки jQuery UI мы используем метод `sortable()` этой библиотеки для преобразования области `cards` в сортируемую область. Все дочерние элементы становятся сортируемыми. Присмотритесь повнимательнее к строке 8; это строка делает все то, что мы делали со встроенной поддержкой Drag and Drop. Конечно, наряду с добавлением этой строки нам пришлось задействовать внешнюю библиотеку.

jQuery UI содержит массу полезных возможностей, от виджетов выбора даты до сложных анимаций. Это огромная библиотека, и возможно, все ее возможности вам не понадобятся. В нашем примере jQuery UI загружается из сети доставки контента. В окончательной версии следует создать специализированную загрузку и включить только те компоненты, которые действительно необходимы. Необходимые инструменты доступны на сайте jQuery UI[1].

Впрочем, даже с обходным решением необходимо обратить внимание на еще одну сторону нашего решения: доступность. В спецификации ничего не сказано о том, что делать людям, которые не могут пользоваться мышью. Если мы реализуем функциональность перетаскивания в своем интерфейсе, нужно будет разработать вспомогательный метод, для работы которого не потребуется ни JavaScript, ни мышь, причем этот метод будет полностью зависеть от того, что мы пытаемся сделать. Построенное нами приложение абсолютно бесполезно для пользователя с ограниченными зрительными возможностями — потому, что доступность не была запланирована с самого начала. Для улучшения доступности можно было бы добавить в каждую карточку поле, определяющее ее положение в общем порядке, и предоставить пользователю возможность ввести число, чтобы последующее нажатие кнопки изменяло порядок карточек. Такой интерфейс даже может оказаться более удобным для пользователей с нормальным зрением, которым просто не нравится идея перетаскивания элементов.

[1] http://jqueryui.com/download/

Как правило, когда речь заходит о реализации технологий, не ищите обходные решения только для пользователей старых браузеров. Думайте об интересах всех пользователей, желающих извлечь максимум пользы из созданного вами продукта или услуги.

Перспективы

Технологии, описанные в этой главе, формально не являются частью HTML5, однако они представляют будущее веб-разработки. Со временем все большая часть функциональности будет перемещаться на сторону клиента. Расширенное управление историей просмотра сделает клиентские приложения и Ajax-приложения более понятными и логичными. Такие сайты, как GitHub и Flickr, уже используют History API, предоставляя обходные решения, чтобы функциональность работала и при отсутствии прямой поддержки. Технология Web Sockets заменяет периодический опрос удаленных служб отображением данных в реальном времени. Теперь, когда протокол относительно стабилизировался, можно ожидать, что технология Web Sockets станет еще более популярной при создании веб-приложений реального времени — особенно сейчас, когда ее поддержка появилась в Internet Explorer 10.

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.

Технология Drag and Drop тоже постепенно развивается. При использовании File API[1] можно создавать приложения, позволяющие пользователям перетаскивать файлы с рабочего стола в браузер. Со временем количество браузеров, поддерживающих File API, вырастет, и вы сможете реализовать возможность отправки файлов посредством перетаскивания без привлечения Flash или Silverlight.

Исследуйте возможности этих технологий и следите за их распространением. Возможно, скоро они начнут играть важную роль в вашем инструментарии веб-разработки.

[1] http://www.w3.org/TR/FileAPI/

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 определения макетов в модели Flexible Box и возможности выдачи Ajax-запросов без ограничений политики единого домена. А если этого покажется недостаточно, вы можете обрабатывать данные в фоновом режиме, отправлять сообщения с сервера клиенту и применять эффекты фильтров к элементам, словно вы работаете в графической программе.

Хотя спецификации еще не устоялись и не получили повсеместной поддержки, инструменты, описанные в этой главе, оказываются весьма полезными в правильных ситуациях. В этой главе рассматриваются следующие темы:

Модель Flexible Box
: Расширенные возможности создания макетов средствами CSS. [C26, F22, S4, O10.6]

Междоменный доступ к ресурсам
: Передача запросов Ajax между доменами. [C4, F3.5, S4, IE10, O12.0, iOS3.2, A2.1]

Web Workers
: Выполнение продолжительных задач в фоновом потоке. [C4, F3.5, S4, IE10, O12.1, iOS5, A2.1]

События на стороне сервера
: Механизм односторонней передачи данных с сервера подключенным клиентам. [C6, F6, S5, O11, iOS4]

Эффекты фильтров CSS [`filter: blur(10px)`]
Применение к элементам различных эффектов: размывка, оттенки серого, сепия, тени и т. д. [C18, S6, O15, iOS6]

3D-графика с использованием WebGL[1]
Вывод 3D-объектов на холсте. [C8, F4, S5.1, O12]

Начнем с рассмотрения существенно усовершенствованного механизма определения макетов в CSS.

11.1. Определение макетов в модели Flexible Box

Одним из самых сложных аспектов использования CSS при формировании макетов является борьба с `float` и `clear` — двумя свойствами CSS, позволяющими создавать привычные нам веб-макеты без использования таблиц.

Однако многие разработчики злоупотребляют `float` и `clear`, поэтому возникла необходимость в более совершенных средствах макетирования. Модель Flexible Box[2] обещает решить эти проблемы, и хотя сейчас спецификация еще не совсем готова, вскоре работа над ней завершится. Новая модель очень, очень заметно упрощает разработку сложных макетов, работающих на любых устройствах.

В этом рецепте модель Flexible Box будет использоваться для создания стандартного макета «боковая панель слева, контент справа», которую мы уже видели тысячи раз. Начнем с базовой разметки страницы HTML.

where_next/flexbox/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Home</title>
    <link rel="stylesheet" href="stylesheets/style.css" />
  </head>
  <body>
    <header>
      <h1>AwesomeCo</h1>
    </header>
```

[1] Может отключаться по умолчанию или требовать установки последних видеодрайверов для нормальной работы.

[2] http://www.w3.org/TR/css3-flexbox/

```
    <div class="container">

      <section id="main">
        <h1>Some Story</h1>

        <p>
          Lorem ipsum dolor sit amet, consectetur adipisicing
          elit, sed do eiusmod tempor incididunt ut labore et
          dolore magna aliqua. Ut enim ad minim veniam, quis
          nostrud exercitation ullamco laboris
          nisi ut aliquip ex ea commodo consequat.
        </p>
        <p>
          Duis aute irure dolor in reprehenderit in voluptate
          velit esse cillum dolore eu fugiat nulla pariatur.
          Excepteur sint occaecat cupidatat non proident, sunt in
          culpa qui officia deserunt mollit anim id est laborum.
        </p>
      </section>
      <section id="sidebar">
        <ul>
          <li><a href="#">Related Link</a></li>
          <li><a href="#">Related Link</a></li>
          <li><a href="#">Related Link</a></li>
          <li><a href="#">Related Link</a></li>
        </ul>
      </section>

    </div>
    <footer>
      <p>Copyright © 2013 AwesomeCo</p>
    </footer>
  </body>
</html>
```

Перед вами типичный шаблон HTML5 со ссылкой на таблицу стилей, `<header>`, `<footer>` и бессмысленным элементом `<div>`, который содержит основную область и боковую панель. Этот элемент `<div>` выполняет функции контейнера. В таблице стилей класс определяется следующим образом:

where_next/flexbox/stylesheets/style.css

```
.container{
  display: -webkit-flex;
```

```
  display: flex;
}
```

Затем мы определяем ширину главной области и используем свойство flex для боковой панели, чтобы она заполняла оставшееся пространство.

where_next/flexbox/stylesheets/style.css

```
#main{
  width: 80%;
  -webkit-order: 2;
          order: 2;
}
#sidebar{
  -webkit-flex: 1;
          flex: 1;

  -webkit-order: 1;
          order: 1;
}
```

Еще больше впечатляет возможность переупорядочения элементов в документе. Главная область определяется в документе первой, но мы можем использовать свойство `order` для перестановки элементов, так что результат выглядит так, как показано на рис. 38.

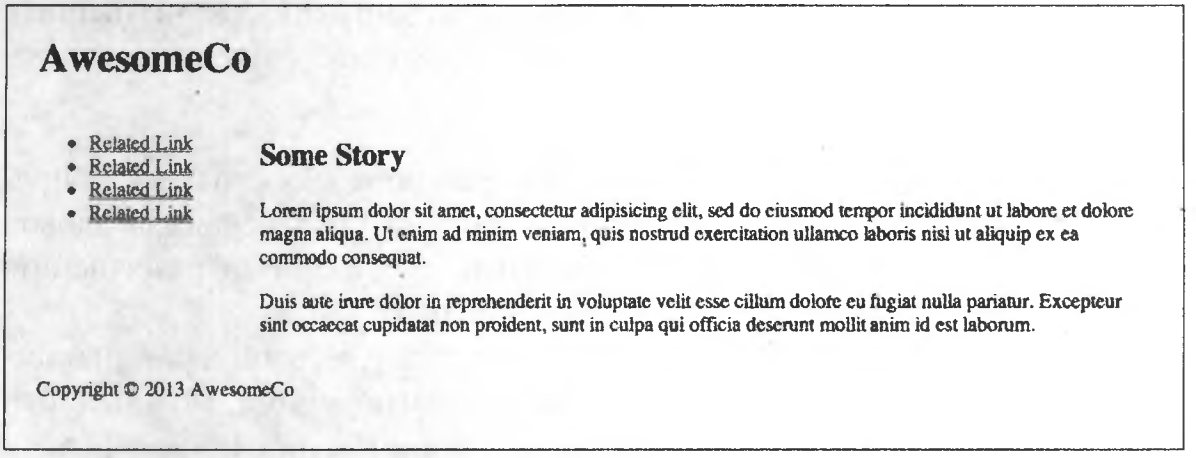


Рис. 38. Боковая панель и главная область находятся там, где мы их разместили

Только представьте, что это означает в контексте динамичного веб-дизайна: используя данную возможность с медиазапросами, вы можете с минимальными усилиями переупорядочивать элементы страницы, существенно упрощая работу пользователя на смартфонах! К сожалению,

поддержка новой модели весьма неоднородна. Internet Explorer 10 и Safari, как и iOS, поддерживают старую версию спецификации. Вы можете организовать условную загрузку таблиц стилей при помощи Modernizr или же воспользоваться Flexie[1], чтобы обеспечить совместимость модели Flexible Box с вашим сайтом. Это может показаться странным, но сегодня это обходное решение используется на многих сайтах. Проанализируйте ситуацию и решите сами, насколько оно подходит для вашего проекта. Безусловно, эта технология выглядит более привлекательно, чем возня с `float` и `clear`.

11.2. Междоменный доступ к ресурсам

Междоменный доступ к ресурсам — мера безопасности, из-за которой обращение с запросами Ajax к странице, размещенной в другом домене, становится практически невозможным. Существуют разные способы обойти это ограничение, но механизм междоменного доступа к ресурсам, или CORS (Cross-Origin Resource Sharing), является стандартным способом передачи запросов между серверами. А самое лучшее, что эта возможность поддерживается практически всеми браузерами, включая Internet Explorer 10. Впрочем, для ее реализации домен, к которому вы пытаетесь обратиться, должен быть настроен для приема запросов CORS, а вы должны настроить свой код для отправки этих запросов. А если говорить конкретнее, сервер должен отвечать следующим заголовком:

```
Access-Control-Allow-Origin: *
```

Все. Если это условие выполнено, у современного браузера не будет проблем с обращением к сервису и обработке запроса. Он отправит заголовок «Origin», а сервер проверит этот заголовок по своей системе разрешений. Если совпадение будет найдено, запрос проходит.

Так как вся эта работа выполняется на сервере, на этом наше знакомство с темой и ограничится. На сайте *http://enable-cors.org/* можно найти другую ценную информацию, включая описание настройки серверов.

11.3. Web Workers

Мы используем JavaScript для всего программирования на стороне клиента, но если выполнение задачи занимает много времени, пользователю приходится дожидаться ее завершения. Иногда это даже приводит к тому,

[1] http://flexiejs.com/

что интерфейс перестает реагировать на действия пользователя. Для решений этой проблемы технология Web Workers создает простой механизм написания параллельных программ или по крайней мере перемещения задач с большой продолжительностью выполнения в фоновые потоки.

Технология Web Workers[1] не входит в спецификацию HTML5, но, возможно, она пригодится вам для организации фоновых вычислений на стороне клиента, так что о ней стоит упомянуть особо.

Рассмотрим простой пример. Мы используем технологию Web Workers для получения данных через общедоступный API YouTube и вывода миниатюры, щелчок на которой воспроизводит видео с YouTube. Примерный результат показан на рис. 39.

Рис. 39. Загрузка видео с использованием Web Workers

Общедоступный API YouTube поддерживает формат JSONP (JavaScript Object Notation with Padding); соответственно вместо создания `XmlHttpRequest`, как это обычно делается, мы присоединяем ключевые слова к URL-адресу вместе с именем функции обратного вызова и добавляем их в тег `<script>` нашего документа. Данные YouTube передаются указанной функции обратного вызова. Нам остается лишь написать функцию обратного вызова для разбора данных. Эта элегантная схема позволяет обойти политику единого домена, поддерживаемую большинством браузеров.

Итак, построим это решение. Начнем с создания довольно стандартной страницы HTML:

where_next/web_workers/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
```

[1] http://www.whatwg.org/specs/web-workers/current-work/

```
    <title>Web Workers</title>
    <style>
      #output > div{float: left; margin-right: 5px;}
    </style>
  </head>
  <body>
    <input type="button" id="button" value="Get Results">
    <div id="output"></div>

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

Страница загружает jQuery и сценарий *scripts/application.js*, обеспечивающий основное взаимодействие со страницей. Также в секцию `<head>` включен небольшой фрагмент CSS, который размещает видеоролики в табличном формате.

Другой файл *javascripts/worker.js* обеспечивает взаимодействие с YouTube. Он станет нашим фоновым потоком, а создается он следующим образом:

where_next/web_workers/javascripts/application.js

```
var worker = new Worker("javascripts/worker.js");
```

Любой файл JavaScript может быть запущен в фоновом режиме, но чтобы его выполнение было действительно независимым, этот сценарий не может обращаться к модели DOM. Это означает, что вы не сможете манипулировать с элементами напрямую, однако данные можно передать фоновому потоку и получить их обратно позднее.

Основной сценарий отправляет сообщения фоновому сценарию методом `postMessage()`:

where_next/web_workers/javascripts/application.js

```
$("#button").click(function(event){
  worker.postMessage("pragprog");
});
```

В данном случае фоновому потоку отправляется условие поиска. Фоновый поток возвращает сообщения, для работы с которыми следует прослушивать событие `onmessage()` рабочего потока:

where_next/web_workers/javascripts/application.js

```
worker.onmessage = function(event){
};

worker.onerror = function(event){
  $("outpout").html("Why do you fail??");
};
```

Этот код выполняется каждый раз, когда фоновый поток отправляет ответное сообщение.

Итак, что же нужно сделать для того, чтобы фоновый поток получал и отправлял сообщения основному приложению? Во-первых, он должен прослушивать событие `onmessage()`. Когда главный сценарий отправляет сообщения фоновому потоку, этот код будет выполняться.

where_next/web_workers/javascripts/worker.js

```
var onmessage = function(event) {
  var query = event.data;
  getYoutubeResults(query);
};
```

Условие поиска передается нашей функции с именем `getYoutubeResults()`, которая конструирует поисковый запрос и отправляет его на YouTube. Обычно при работе с JSONP запрос создается включением в главную страницу тега `<script>`. Но в фоновом потоке модель DOM недоступна, как и окно браузера и т. д. Однако в нашем распоряжении имеется исключительно гибкий метод `importScripts()`, который может обращаться по локальным, относительным и удаленным URL-адресам.

where_next/web_workers/javascripts/worker.js

```
var getYoutubeResults = function(searchTerm) {
  var callback = "processResults";
  url = "http://gdata.youtube.com/feeds/videos?vq=" + searchTerm +
        "&alt=json-in-script&max-results=5&callback=" + callback;
  importScripts(url);
};
```

Этот метод строит URL-адрес поиска с указанием ключевого слова поиска и функции обратного вызова. Нам остается лишь написать функцию обратного вызова с именем `processResults()`.

Ответ YouTube выглядит примерно так (хотя и содержит гораздо больше информации):

```
// Ответ YouTube
processResults({
  "version": "1.0",
  "feed": {
    "title": {
      "$t": "Videos matching: pragprog",
      "type": "text"
    },
    "entry": [{
      "title": {
        "$t": "Using tmux for productive mouse-free programming",
      },
      "media$group": {
        "media$content": [{
          "url": "http://www.youtube.com/v/JXwS7z6Dqic",
          "type": "application/x-shockwave-flash",
          "medium": "video",
          "isDefault": "true"
        }],
        "media$thumbnail": [{
          "url": "http://i.ytimg.com/vi/JXwS7z6Dqic/0.jpg",
          "height": 360,
          "width": 480,
          "time": "00:02:01"
        }]
      }
    }]
  }
});
```

Внимательно присмотритесь к коду ответа — он вызывает `processResults()` с передачей набора данных. Когда наш фоновый сценарий получает такой ответ от YouTube, он должен его выполнить. Мы должны разобрать данные и отправить их пользовательскому интерфейсу. Здесь мы извлечем только миниатюру и ссылку на видео и поместим их в новый объект, который будет отправлен при помощи `postMessage()`:

where_next/web_workers/javascripts/worker.js

```
var processResults = function(json) {
  var data, result;
  for(var index = 0; index < json.feed.entry.length; index++){
    result = json.feed.entry[index]["media$group"];
    data = {
      thumbnail: result["media$thumbnail"][0]["url"],
      videolink: result["media$content"][0]["url"]
```

```
    }
    postMessage(data);
  }
};
```

Наконец, в файле *javascripts/application.js* мы заполняем обработчик события `onmessage()` и добавляем каждый результат к странице:

where_next/web_workers/javascripts/application.js

```
worker.onmessage = function(event){

var img = $("<img>");
var link = $("<a>");
var result = event.data;
var wrapper;

link.attr("href", result.videolink);
img.attr("src", result.thumbnail);
link.append(img);
wrapper = link.wrap("<div>").parent();
$("#output").append(wrapper);

};
worker.onerror = function(event){
  $("outpout").html("Why do you fail??");
};
```

Возможно, вы заметили, что API Web Workers очень похож на API междоменной передачи сообщений (см. рецепт 31). Мы получаем сообщение от фонового потока и отвечаем на него. К сожалению, в Internet Explorer версий ниже 10 технология Web Workers не поддерживается. Но если вам потребуется выполнить на стороне клиента более серьезную работу без блокировки, эту тему стоит изучить подробнее. В этой конкретной ситуации можно проверить поддержку Web Workers, и если она не обнаружена, выдать обычный запрос JSONP средствами jQuery.

Отладка решений с Web Workers может быть весьма нетривиальной. Поскольку мы не можем легко обратиться к DOM и не можем использовать `console.log()`, наши возможности ограничиваются выдачей исключений или использованием `postMessage()` для возвращения данных, которые затем выводятся на странице из обработчика `onmessage()`. Эти способы выглядят неуклюже, но работают.

Технология Web Workers отлично подходит для ситуаций с выполнением продолжительных операций, часто требующих заметных вычислительных мощностей, которые должны выполняться без блокировки главного

потока пользовательского интерфейса. Если ваше приложение занимается обработкой данных на стороне клиента, эту тему стоит изучить подробнее.

11.4. События на стороне сервера

Веб-сокеты хороши, но они требуют использования еще одного протокола и скорее предназначены для двусторонних коммуникаций. Если вам всего лишь нужно передать данные с сервера клиенту, используйте технологию SSE (Server-Sent Events), которая работает на базе традиционного протокола HTTP.

Для демонстрации мы создадим очень простую страницу для вывода сообщений с сервера. Веб-сервер, включенный в архив примеров, уже поддерживает SSE, так что мы сосредоточимся на реализации на стороне клиента. Создайте простую страницу HTML с местом для вывода сообщений:

where_next/html5_sse/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>AwesomeCo Messages</title>
    <link rel="stylesheet" href="stylesheets/style.css">
  </head>
  <body>
    <h2>AwesomeCo Messages</h2>
    <div id="message">connecting....</div>
    <script src='javascripts/streamer.js'></script>
  </body>
</html>
```

Также нам понадобится место для размещения кода JavaScript. Создайте файл *javascripts/streamer.js* и включите ссылку на него в файл HTML:

where_next/html5_sse/index.html

```
<script src='javascripts/streamer.js'></script>
```

Переходим к выводу данных на странице.

Прослушивание событий

В файл *javascripts/streamer.js* включается функция `createMessageListeners()`, которая определяет все обработчики для событий на стороне сервера и устанавливает связь с потоком событий сервера.

where_next/html5_sse/javascripts/streamer.js

```
var createMessageListeners = function(){
  var messageSource = new EventSource("/stream");
}
```

Для создания связи с потоком событий может использоваться объект `EventSource`. В зависимости от браузера может потребоваться, чтобы он находился на одном сервере с веб-страницей из-за ограничений политики единого домена. Поддержка CORS для реализации SSE еще не получила повсеместного распространения.

```
createMessageListeners();
```

После установления подключения сервер начинает отправлять непрерывный поток сообщений. Простейшее сообщение выглядит примерно так:

```
data: We are bringing even more awesomeness to you!
```

Оно начинается со слова `data`, за которым следует двоеточие, а затем текст сообщения с последующей пустой строкой — не просто символом новой строки, а совершенно пустой строкой. Сообщение вида:

```
data: We are bringing even more awesomeness to you!
data: Are you ready to be even more awesome?
```

интерпретируется как одно сообщение, состоящее из нескольких строк. С другой стороны, фрагмент

```
data: We are bringing even more awesomeness to you!

data: Are you ready to be even more awesome?
```

содержит два разных сообщения. Данные передаются в формате простого текста или в формате JSON, который разбирается на стороне клиента. Протокол очень простой, но мощный. Сервер продолжает отправлять сообщения всем подключенным клиентам вплоть до своей остановки или отключения всех клиентов.

Чтобы получить сообщение, мы обрабатываем событие `message` класса `EventSource` и извлекаем сообщение из `event.data`:

where_next/html5_sse/javascripts/streamer.js

```
messageSource.addEventListener("message", function(event){
  document.getElementById("message").innerHTML = event.data;
}, false);
```

Вот и все, что требуется. Основная работа связана с созданием сервера и определением того, как должны обрабатываться сообщения после их получения.

Вы можете прослушивать другие события (такие, как `close`, `open` и даже определяемые вами пользовательские событий). Если сервер отправляет клиенту сообщение вида:

```
event: stockupdate
data: {"stock": "MSFT", "value": "34.01"}
```

вы должны прослушивать события `stockupdate` вместо событий `message` и извлечь из них данные. Вы даже можете сообщить клиенту продолжительность ожидания повторной попытки, отправив ему с сервера сообщение:

```
retry: 10000
```

В этом случае клиент будет ожидать десять секунд между запросами вместо трех секунд, используемых по умолчанию.

Это решение может работать в старых браузерах, не поддерживающих события на стороне сервера. При необходимости используйте Modernizr для проверки поддержки и загрузите простое решение EventSource[1].

Реализация собственного сервера

Чтобы проделать все это в продуктивной среде, необходимо написать серверный код для отправки сообщений в указанном формате. Сервер должен отправлять потоковые ответы только в том случае, если клиент запросит их при помощи следующего заголовка:

```
Accept: text/event-stream
```

А еще сервер должен обязательно задать заголовки ответа, идентифицирующие его как потоковый ответ:

```
Content-Type: text/event-stream;charset=UTF-8
Cache-Control: no-cache
Connection: keep-alive
```

Когда настройка ответов будет завершена, переходите к отправке простого текста сообщений всем подключенным клиентам. И конечно, удалите из рассылки всех клиентов, которые прекратили прослушивание. Пример

[1] https://github.com/remy/polyfills/blob/master/EventSource.js

очень примитивного сервера приведен в файле *lib/sse.js* из кода примеров книги; вы также можете построить сервер на базе реализации node-sse[1], которая берет на себя очень многие технические подробности.

11.5. Эффекты фильтров

Эффекты фильтров CSS позволяют выполнять графические операции с элементами при вставке их в документ[2]: гауссову размывку, отражения, слияние, композицию и многие другие эфы, для реализации которых традиционно использовались графические редакторы.

До окончательной версии спецификации еще довольно далеко, и пока лишь часть этих фильтров поддерживается в браузерах на базе WebKit (таких, как Chrome и Safari). А если говорить конкретно, в WebKit-браузерах могут применяться следующие эффекты фильтров:

`blur() blur(10px);`

Размывка изображения на заданное расстояние в пикселах.

`grayscale() grayscale(0.5);`

Подавление цветов в изображении по шкале от 0 до 1, где 0 соответствует полному цвету, а 1 — выводу исключительно в оттенках серого. Значение также может задаваться в процентах.

`drop-shadow() drop-shadow(5px 5px 5px #333)`

Имитация тени, отбрасываемой изображением.

`sepia() sepia(0.5);`

Имитация сепии (эффект коричневатого оттенка старых фотографий) по шкале от 0 до 1, где 0 — обычное изображение, а 1 — полное преобразование графики в тональность сепии. Значение также может задаваться в процентах.

`brightness() brightness(1.0);`

Яркость цвета элемента: 0 – элемент полностью темный, 1 – нормальный, 10 — максимальная яркость. Значение также может задаваться в процентах, 100 % соответствуют нормальному уровню яркости.

[1] https://npmjs.org/package/sse

[2] http://www.w3.org/TR/2013/WD-filter-effects-20130523/

`contrast() contrast(1.0);`

Регулировка контраста, то есть различия между темными и светлыми участками в цвете элемента: 0 – контраст отсутствует, 1 – нормальный контраст, 10 – максимальный контраст. Значение также может задаваться в процентах, 100 % соответствуют нормальному уровню контраста, а все превышающие значения усиливают контрастность.

`hue-rotate() hue-rotate(90deg);`

Поворот оттенка элемента по цветовому колесу на заданный угол в градусах или радианах.

`saturate() saturate(0.5);`

Управление насыщенностью изображения по шкале от 0 до 1, где 0 соответствует отсутствию насыщенности, а 1.0 — полной насыщенности. Значение также может задаваться в процентах.

`invert() invert(1);`

Инверсия цветов с созданием эффекта «негатива» по шкале от 0 до 1 или от 0 до 100 %. Серое изображение соответствует середине шкалы.

`opacity() opacity(1);`

Определение степени прозрачности элемента, чтобы были видны находящиеся за ним другие элементы или цвета. Значения задаются по шкале от 0 до 1 или от 0 до 100 %. В точке 0 элемент становится полностью невидимым, а в точке 1 он полностью непрозрачен.

Самое замечательное заключается в том, что все эти свойства также работают с переходами и анимациями. Например, если вы хотите, чтобы изображения на странице выводились в оттенках серого до момента наведения на них указателя мыши, желаемый эффект реализуется крошечным фрагментом CSS:

where_next/filters/stylesheets/style.css

```
img.photo{
  -webkit-filter: grayscale(1);
  -webkit-transition: -webkit-filter 0.5s linear;
}

img.photo:hover{
  -webkit-filter: none;
}
```

Такое решение реализуется гораздо проще, чем создание двух версий графики и их переключение средствами JavaScript или CSS.

Конечно, эти возможности поддерживаются еще не всеми браузерами. И хотя их можно заставить работать даже без встроенной поддержки, вероятно, лучше этого не делать — или использовать, но не беспокоиться об обходных решениях. Когда поддержка фильтров стабилизируется, а эффекты начнут поддерживаться другими браузерами, вы сможете применять их там, где это разумно. Но как и все визуальные эффекты, их следует применять осмотрительно. Не «перенасыщайте» страницу множеством фильтров, отвлекающих от содержимого.

11.6. 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[1] 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[2].

11.7. Вперед!

Разработчиков ждут интересные времена. Эта книга дает лишь общее представление о ближайших тенденциях веб-разработки. В спецификациях содержится намного больше полезной информации и с ними определенно стоит познакомиться поближе. С течением времени спецификации будут изменяться, браузеры будут обретать новую функциональность, а перед вами откроются новые возможности. 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!

---

[1] http://www.khronos.org/registry/webgl/specs/latest/

[2] http://learningwebgl.com/blog/

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: Chrome, F: Firefox, S: Safari, IE: Internet Explorer, O: Opera, IOS: устройства iOS с Safari, A: Android Browser.

A.1. Новые элементы

См. рецепт 1, «Реструктуризация блога с использованием семантической разметки», с. 33.

`<header>`
: Определение заголовка страницы или раздела. [C5, F3.6, S4, IE8, O10]

`<footer>`
: Определение завершителя страницы или раздела. [C5, F3.6, S4, IE8, O10]

`<nav>`
: Определение области навигации страницы или раздела. [C5, F3.6, S4, IE8, O10]

`<section>`
: Определение логической области страницы или группировка контента. [C5, F3.6, S4, IE8, O10]

`<article>`
: Определение статьи (логически завершенного блока контента). [C5, F3.6, S4, IE8, O10]

`<aside>`
: Определение вторичного или связанного контента. [C5, F3.6, S4, IE8, O10]

Другие элементы:

Списки описаний
: Определение списка имен и связанных с ними значений (таких, как определения или описания). [Все браузеры]

  См. рецепт 4, «Применение списков описаний для определения FAQ», с. 56.

`<meter>`
: Представление величины, находящейся в заданном диапазоне. [C8, F16, S6, O11]

  См. рецепт 2, «Вывод информации о ходе выполнения операции с использованием элемента <meter>», с. 46.

`<progress>`
: Отображение информации о ходе некоторой операции в реальном времени. [C8, F6, S6, IE10, O11]

  См. рецепт 2, «Вывод информации о ходе выполнения операции с использованием элемента <meter>», с. 46.

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 3, «Создание всплывающих окон с пользовательскими атрибутами данных», с. 51.

Редактирование «на месте» `<p contenteditable>lorem ipsum</p>`
: Поддержка редактирования контента «на месте» в браузере. [C4, S3.5, S3.2, IE6, O10.1, iOS5, A3]

  См. рецепт 9, «Редактирование „на месте“», с. 84.

A.3. Формы

См. рецепт 5, «Описание данных при помощи новых полей», с. 61.

`<input type="email">`
: Поле для ввода адреса электронной почты. [O10.1, IOS, A3]

`<input type="url">`
Поле для ввода URL-адреса. [O10.1, IOS, A3]

`<input type="range">`
Ползунок. [C5, S4, F2.3, IE10, O10.1]

`<input type="number">`
Поле для ввода числовых данных, часто в виде элемента-счетчика. [C5, S5, O10.1, IOS5, A3]

`<input type="color">`
Поле для выбора цвета. [C5, O11]

`<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`. [S5, O10.1]

`<input type="search">`
Поле для ввода ключевых слов поиска. [C5, S4, O10.1, IOS]

A.4. Атрибуты полей форм

`<input type="text" autofocus>`
Поддержка передачи фокуса конкретному элементу формы. [C5, S4]

См. рецепт 6, «Использование автофокуса для перехода к первому полю», с. 73.

`<input type="email" placeholder="me@example.com">`
Поддержка автоматического включения текста в поле формы. [C5, S4, F4]

См. рецепт 7, «Заполняющий текст», с. 74.

`required` [`<input type="email" required >`]
Блокировка отправки страницы, если поле осталось незаполненным. [C23, F16, IE10, O12]

См. рецепт 8, «Проверка пользовательского ввода без использования JavaScript», с. 78.

`pattern` [`<input type="text" pattern="^[1-9]+[0-9]*$">`]
Блокировка отправки страницы, если содержимое поля не соответствует заданному образцу. [C23, F16, IE10, O12]

См. раздел 8, «Проверка пользовательского ввода без использования JavaScript», с. 78.

A.5. Доступность

Атрибут `role` [`<div role="document">`]
: Описание обязанностей элементов для экранных дикторов. [C3, F3.6, S4, IE8, O9.6]

  См. рецепт 14, «Роли ARIA и упрощение навигации», с. 121.

`aria-live` [`<div aria-live="polite">`]
: Идентификация автоматически обновляемых (вероятно, средствами Ajax) областей. [F3.6 (Windows), S4, IE8]

  См. рецепт 15, «Создание обновляемых областей с улучшенной доступностью», с. 126.

`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 15, «Создание обновляемых областей с улучшенной доступностью», с. 126.

`<scope>` [`<th scope="col">Time</th>`]
: Установление связи заголовка таблицы со столбцами или строками таблицы. [Все браузеры]

  См. рецепт 16, «Улучшение доступности таблиц», с. 133.

`<caption>` [`<caption>This is a caption</caption>`]
: Создание подписи для таблицы. [Все браузеры]

  См. рецепт 16, «Улучшение доступности таблиц», с. 133.

`aria-describedby` [`<table aria-describedby="summary">`]
: Установление связи описания с элементом. [F3.6 (Windows), S4, IE8]

  См. рецепт 16, «Улучшение доступности таблиц», с. 133.

A.6. Мультимедиа

`<canvas>` [`<canvas><p>Alternative content</p></canvas>`]
: Создание растровой графики из кода JavaScript. [C4, F3, S3.2, IE9, O10.1, IOS3.2, A2]

  См. рецепт 17, «Рисование логотипа», с. 141.

`<svg>` [`<svg><!-- XML content --></svg>`]
: Создание векторной графики в разметке XML. [C4, F3, S3.2, IE9, O10.1, iOS3.2, A2]

  См. рецепт 19, «Создание векторной графики SVG», с. 157.

`<audio>` [`<audio src="drums.mp3"></audio>`]
Воспроизведение аудио встроенными средствами браузера. [C4, F3.6, S3.2, IE9, O10.1, IOS3, A2]

См. рецепт 20, «Работа с аудио», с. 169.

`<video>` [`<video src="tutorial.m4v"></video>`]
Воспроизведение видео встроенными средствами браузера. [C4, F3.6, S3.2, IE9, O10.5, IOS3, A2]

См. рецепт 21, «Внедрение видео», с. 174.

A.7. CSS3

`: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 10, «Стилевое оформление таблиц с использованием псевдоклассов», с. 95.

`:first-child` [`p:first-child{color:blue;}`]
Поиск первого дочернего элемента. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 10, «Стилевое оформление таблиц с использованием псевдоклассов», с. 95.

`: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 10, «Стилевое оформление таблиц с использованием псевдоклассов», с. 95.

`:last-child` [`p:last-child{color:blue;}`]
Поиск последнего дочернего элемента. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 10, «Стилевое оформление таблиц с использованием псевдоклассов», с. 95.

`: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 10, «Стилевое оформление таблиц с использованием псевдоклассов», с. 95.

`: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 10, «Стилевое оформление таблиц с использованием псевдоклассов», с. 95.

`:last-of-type` [`p:last-of-type{color:blue;}`]
Поиск последнего элемента заданного типа. [C2, F3.5, S3, IE9, O9.5, IOS3, A2]

См. рецепт 10, «Стилевое оформление таблиц с использованием псевдоклассов», с. 95.

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 13, «Создание многостолбцовых макетов», с. 112.

`: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 11, «Печать ссылок (:after)», с. 105.

Медиазапросы [`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 12, «Построение мобильных интерфейсов», с. 108.

`border-radius` [`border-radius: 10px;`]
Закругление углов элементов. [C4, F3, S3.2, IE9, O10.5]

См. рецепт 23, «Закругление прямых углов», с. 187.

Поддержка RGBa [`background-color: rgba(255,0,0,0.5);`]
Использование цветов RGB вместо шестнадцатеричных кодов (с альфа-каналом). [C4, F3.5, S3.2, IE9, O10.1]

См. рецепт 24, «Тени, градиенты и преобразования», с. 192.

`box-shadow` [`box-shadow: 10px 10px 5px #333;`]
Создание теней, отбрасываемых элементами. [C3, F3.5, S3.2, IE9, O10.5]

См. рецепт 24, «Тени, градиенты и преобразования», с. 192.

Повороты [`transform: rotate(7.5deg);`]
Поворот любого элемента. [C3, F3.5, IE9, S3.2, O10.5]

См. рецепт 24, «Тени, градиенты и преобразования», с. 192.

Градиенты [`linear-gradient(top, #fff, #efefef);`]
Построение градиента для использования в качестве графического изображения. [C4, F3.5, S4]

См. рецепт 24, «Тени, градиенты и преобразования», с. 192.

`src: url(http://example.com/awesomeco.ttf); font-weight: bold; }]`
Возможность использования конкретных шрифтов в CSS. [C4, F3.5, S3.2, IE5, O10.1]

См. рецепт 25, «Использование шрифтов», с. 200.

Переходы [`transition: background 0.3s ease`]
Плавный переход свойства CSS от одного значения к другому. [C4, F3.5, S4, IE10]

См. рецепт 26, «Переходы и анимации», с. 206.

Анимации [`animation: shake 0.5s 1;`]
Плавный переход свойства CSS от одного значения к другому посредством анимации по ключевым кадрам. [C4, F3.5, S4, IE10]

См. рецепт 26, «Переходы и анимации», с. 206.

Эффекты фильтров CSS [`filter: blur(10px) `]
Применение к элементам различных эффектов: размывка, оттенки серого, сепия, тени и т. д. [C18, S6, O15, iOS6]

См. раздел 11.5, «Эффекты фильтров», с. 295.

Модель Flexible Box
Расширенные средства CSS для построения макетов. [C26, F22, S4, O10.6]

См. раздел 11.1, «Определение макетов в модели Flexible Box», с. 283.

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 27, «Сохранение настроек с использованием Web Storage», с. 221.

`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, IOS3.2, A]

См. рецепт 27, «Сохранение настроек с использованием Web Storage», с. 221.

IndexedDB

Хранилище объектов, содержимое которого сохраняется между сеансами. [C25, F10, IE10]

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. Технология исключена из активной спецификации. [C5, S3.2, O10.5, IOS3.2, A2]

См. врезку «Web SQL Databases», с. 241.

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 29, «Автономная работа», с. 242.

History

Управление историей просмотра. [C5, S4, IE8, F3, O10.1 IOS3.2, A2]

См. рецепт 30, «История просмотра», с. 248.

Cross-Document Messaging

Передача сообщений между окнами с контентом, загруженным в разных доменах. [C5, S5, F4, IOS4.1, A2]

См. рецепт 31, «Передача информации между доменами», с. 253.

Web Sockets

Создание подключения с поддержкой состояния между браузером и сервером. [C5, S5, F4, IOS4.2]

См. рецепт 32, «Чат на базе Web Sockets», с. 260.

Geolocation

Получение информации о широте и долготе. [C5, S5, F3.5, O10.6, IOS3.2, A2]

См. рецепт 33, «Определение местоположения: Geolocation», с. 269.

Drag-and-Drop

Поддержка перетаскивания. [C4, F3.5, S4, IE6, A2]

См. раздел 34, «Перетаскивание», с. 273.

Междоменный доступ к ресурсам

Передача запросов Ajax между доменами. [C4, S4, F3.5, IE10, O12.0, iOS3.2, A2.1]

Web Workers

Выполнение продолжительных задач в фоновых потоках. [C4, S4, F3.5, IE10, O12.1, iOS5, A2.1]

См. раздел 11.3, «Web Workers», с. 286.

События на стороне сервера

Механизм односторонней передачи данных с сервера подключенным клиентам. [C6, F6, S5, O11, iOS4]

См. раздел 11.4, «События на стороне сервера», с. 292.

3D-графика с использованием WebGL[1]

Вывод 3D-объектов на холсте. [C8, F4, S5.1, O12]

См. раздел 11.6, «11.6 WebGL», с. 297.

[1] Может отключаться по умолчанию или требовать установки последних видеодрайверов для нормальной работы.

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 как простых обходных решений, так и сложных веб-приложений.

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.

В jQuery задачи манипуляции с моделью DOM и обработки событий решаются предельно просто. jQuery использует селекторы CSS для поиска элементов, упаковывает эти элементы в специальный объект и позволяет разработчику изменять эти элементы или добавлять к ним обработчики событий. Все, что делает jQuery, может быть сделано в обычном коде JavaScript, но у jQuery есть преимущество: библиотека сама решает проблемы межбраузерной совместимости и делает синтаксис более универсальным.

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:

---

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

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

jqueryprimer/simple_selection.html

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

Количество одновременных подключений к серверу из браузера ограничено. Если распределить графику и сценарии по нескольким серверам, 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");
```

Точно такая же запись используется для поиска всех тегов `<h1>` на странице. Если потребуется найти все элементы с классом `important`, используйте запись

```
$(".important");
```

Еще раз посмотрите на эти два примера. Они различаются только используемым селектором CSS. Поиск элементов с идентификатором `header` осуществляется простой командой

```
var header = $("#header");
```

А если бы нам потребовалось найти все ссылки на боковой панели, мы бы использовали для этого соответствующий селектор `$("#sidebar a")`.

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:

jqueryprimer/simple_selection.html

```
$("h1").hide();
```

Метод `hide()` часто используется в книге для сокрытия разделов страниц, которые должны отображаться только при отключении JavaScript (например, расшифровок аудироликов или другого обходного контента). Для отображения скрытых элементов достаточно вызвать метод `show()`.

Для простого переключения видимости элемента в противоположное состояние достаточно вызвать `toggle()`.

Если функция jQuery находит несколько элементов, соответствующих заданному селектору, то все найденные элементы будут скрыты/отображены/переключены. Таким образом, предыдущий пример скрывает не первый элемент `<h1>`, а все найденные элементы!

jqueryprimer/simple_selection.html

```
$("#sidebar a").hide();
```

Методы html, val и attr

Метод `html()` используется для чтения и записи внутреннего контента заданного элемента.

jqueryprimer/methods.html

```
$("#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.

jqueryprimer/methods.html

```
<form id="task_form">
  <label for="task">Task</label>
  <input type="text" id="task" >
  <input type="submit" value="Add">
</form>
<ul id="tasks">
</ul>
```

Элементы списка можно создавать, присоединяя их при отправке формы.

jqueryprimer/methods.html

```
$(function(){
  $("#task_form").submit(function(event){
    event.preventDefault();
    var new_element = $("<li>" + $("#task").val() + "</li>");
    $("#tasks").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.

jqueryprimer/methods.html

```
var wrapper = $("#message").wrap("<div class='wrapper'></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 хешей JavaScript для применения к элементу сразу нескольких правил CSS:

jqueryprimer/methods.html

```
$("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:

jqueryprimer/methods.html

```
$("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.

jqueryprimer/simple_selection.html

```
$("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()`. Данная возможность особенно полезна в том случае, когда новые элементы нужно дополнить обработчиками событий или иным поведением.

jqueryprimer/create_elements.html

```
var input = $("input");
```

Хотя того же результата можно добиться вызовом `document.createElement("input");`, использование функции jQuery упрощает дополнительные вызовы методов.

jqueryprimer/create_elements.html

```
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.

Также время от времени возникает необходимость в удалении элементов из DOM. После того как элемент будет выбран функцией jQuery, мы можем легко удалить всех его потомков:

jqueryprimer/remove_elements.html

```
$("#animals").empty();
```

или же удалить сам элемент:

jqueryprimer/remove_elements.html

```
$("#animals").remove();
```

Все это бывает весьма полезно при построении динамичных веб-приложений.

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:

jqueryprimer/popup.html

```
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 `on()`. Например, при реализации перетаскивания из спецификации HTML5 необходимо отменить событие `ondragover`. Метод `on()` используется следующим образом.

jqueryprimer/events.html

```
target = $("#droparea")
target.bind('dragover', function(event) {
  if (event.preventDefault) event.preventDefault();
  return false;
});
```

Обратите внимание: для обрабатываемого события префикс `on` не указывается.

Привязка событий на более высоком уровне

Добавление событий к отдельным элементам замедляет вывод страницы и увеличивает потребление ресурсов. Вместо добавления событий на уровне отдельных элементов мы можем прослушивать события элемента, который их содержит:

jqueryprimer/events.html

```
1  var links = $("#links");
2  links.click(function(event){
3    link = event.target;
4    var address = $(link).attr('href');
5    event.preventDefault();
6    window.open(address);
7  });
```

В строке 3 свойство `event.target` используется для получения ссылки на фактический элемент, на котором был сделан щелчок. Полученный элемент упаковывается в объект jQuery для дальнейших операций. Этот способ работает даже при динамическом добавлении новых элементов в указанную область.

Иногда отслеживание событий должно быть более конкретным. В таких ситуациях можно определить обработчик события для родителя, но при этом указать, к каким элементам он должен применяться:

```
links.on("click","a" , function(event){
element = $(this);
  // Код
});
```

Селектор передается во втором аргументе `on()`. Результат получается тем же, но с большей гибкостью: если класс содержит множество элементов, для которых отслеживаются разные события, этот способ позволит легко установить все необходимые связи.

Исходное событие

При использовании событийных функций jQuery (таких, как `on()` или `click()`) jQuery упаковывает событие JavaScript в отдельный объект, 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).on("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.

Недостаток такого подхода заключается в том, что JavaScript «не видит» никакие элементы документа до их объявления. Если поместить код JavaScript в секцию `<head>` документа, то он будет немедленно выполнен; при этом элементы, с которыми он должен взаимодействовать, будут недоступны, поскольку браузер их еще не построил. 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:

jqueryprimer/ready.html

```
$(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");
});
```

Паттерн с загрузкой сценариев методом `document.ready()` чрезвычайно популярен в приложениях JavaScript. Однако часто можно обойтись без него, размещая вызовы внешних сценариев в конце страницы перед закрывающим тегом `<body>`. При отсутствии сценариев, асинхронно изменяющих DOM, беспокоиться не о чем. Но если вы не контролируете по-

рядок загрузки сценариев, а также при создании элементов «на ходу», использование `document.ready()` будет правильным решением.

Б.7. Разумное использование jQuery

jQuery — большая библиотека, которая не всегда является необходимой. Если в вашей странице требуется всего лишь найти элемент по идентификатору, ограничьтесь записью

```
navbar = document.getElementById("#navbar");
```

Загружать всю библиотеку jQuery для одного-двух применений было бы неэффективно. Современные браузеры позволяют реализовать большую часть функциональности jQuery такими методами, как `document.querySelector()` и `document.querySelectorAll()`, которые используются в современных браузерах для выбора элементов с использованием синтаксиса селекторов CSS:

```
links = document.querySelectorAll("a.popup");
```

Используйте jQuery тогда, когда эта библиотека упростит вашу работу, но не бойтесь отказываться от нее там, где такое решение выглядит более разумно.

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.

Кодирование аудио и видео

B

Кодирование аудио- и видеоматериалов для использования с тегами HTML5 `<audio>` и `<video>` — выходящая за рамки книги тема. И все же 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[1]. 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[2]. 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[3]. Там представлена исключительно полезная

---

[1] http://lame.sourceforge.net/

[2] http://wiki.xiph.org/Vorbis-tools

[3] 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

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.

Кодирование видео в несколько форматов занимает много времени; обязательно перепроверьте полноту и правильность видеоматериалов, прежде чем нажимать кнопку Encode. По возможности закодируйте небольшие фрагменты видеоматериалов и протестируйте их, чтобы подобрать оптимальные настройки.

---

[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/

Ресурсы

Apple — HTML5: *http://www.apple.com/html5/*
: Страница фирмы Apple о поддержке HTML5 и веб-стандартов в браузере Safari.

Can I use...: *http://caniuse.com/*
: Таблицы совместимости браузеров для HTML5, CSS3 и других сопутствующих технологий.

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.

HTML5 Cross Browser Polyfills: *https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills*
: Обширный список решений, реализующих функциональность HTML5 и CSS3 в неподдерживаемых браузерах.

HTML5 Please: *http://html5please.com/*
: Краткие рекомендации по поводу того, какими возможностями HTML5 уже можно пользоваться, а от каких лучше держаться подальше.

Internet Explorer 10 — руководство для разработчиков:
http://msdn.microsoft.com/library/ie/hh673549.aspx
: Подробная информация о возможностях HTML5, CSS3 и JavaScript, поддерживаемых Internet Explorer 10 и приложениями Windows Store.

Microsoft IE Test-Drive: *http://ie.microsoft.com/testdrive/*

Демонстрация возможностей HTML5 (и другой сопутствующей функциональности) в Internet Explorer 10.

Kira's Web Toolbox: «Создание файла политики сокетов Flash»: *http://www.lightsphere.com/dev/articles/flash_socket_policy.html*

Подробное описание файлов политики сокетов Flash, необходимых для разработки обходных решений Web Socket.

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.

Video.js: *http://videojs.com*

Библиотека JavaScript, упрощающая воспроизведение видео HTML5.

WebPlatform.org: *http://webplatform.org/*

Подробная документация и учебники по HTML5, CSS, JavaScript и сопутствующим технологиям.

Библиография

[Bur12] Trevor Burnham. *Async JavaScript*. The Pragmatic Bookshelf, Raleigh, NC and Dallas, TX, 2012.

[Duc1 1] Jon Duckett. *HTML and CSS: Design and Build Websites*. John Wiley & Sons, New York, NY, 2-11.

[HT00] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Reading, MA, 2000.

[Por10] Christophe Porteneuve. *Pragmatic Guide to JavaScript*. The Pragmatic Bookshelf, Raleigh, NC and Dallas, TX, 2010.

[Zel09] Jeffrey Zeldman. *Designing with Web Standards*. New Riders Press, Upper Saddle River, NJ, 2009.