Марк Пилгрим

bhv

ПОГРУЖЕНИЕ В HTML5

- Новые семантические элементы <header>, <footer>, <section> и др.
- Новый элемент для работы с двумерной графикой <Canvas>
- Новые типы полей ввода для Web-форм
- Возможности встраивания видео в Web-страницы без использования дополнительных плагинов
- Возможности построения Web-приложений, способных работать в автономном режиме при отсутствии подключения к Интернету
- Возможности геопозиционирования
- Преимущества использования локального хранилища по сравнению с обычными cookie-файлами
- Создание собственных словарей микроданных для настройки разметки, используемой поисковиками при выводе результатов поиска

HTML5: Up and Running

Mark Pilgrim

Марк Пилгрим

ПОГРУЖЕНИЕ В HTML5

Санкт-Петербург

«БХВ-Петербург»

2011

УДК 681.3.06
ББК 32.973.26-018.2
П32

Пилгрим М.

П32 Погружение в HTML5: перев. с англ. — СПб.: БХВ-Петербург, 2011. — 304 с.: ил.

ISBN 978-5-9775-0688-5

Подробное руководство по всем новшествам стандарта HTML5. Показано, как использовать в Web-разработках новые функциональные возможности, открывающиеся при применении HTML5. Представлено множество простых и понятных практических примеров, позволяющих использовать разметку HTML5 для добавления графики, аудио, видео, автономных возможностей и много другого.

Для Web-разработчиков

УДК 681.3.06
ББК 32.973.26-018.2

Эта книга написана на основе первоисточника, находящегося по адресу http://diveintohtml5.org и поддерживаемого автором

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Перевод с английского и редактирование	*Ольги Кокоревой*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 20.04.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 24,51.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12.

ISBN 978-5-9775-0688-5

Оглавление

Благодарности

Автор приносит глубокую благодарность издательству O'Reilly Media, опубликовавшего версию этой книги, защищенную их авторскими правами, но любезно предоставившего авторую разрешение опубликовать ее на своем сайте на условиях лицензии Creative Commons Attribution 3.0 Unported (см. **http://diveintohtml5.org/about.html**).

Кроме того, в книге использованы иллюстрации из Open Clip Art Library (**http://openclipart.org/**) и других источников, владельцам которых я признателен за разрешение использовать в своей книге результаты их труда.

Далее, в книге использован код `jQuery` (MIT), `Modernizr` (MIT), `geo_location_javascript.js` (MIT), `gears_init.js` (BSD), `highlighter.js` (ASL 2.0), `excanvas.js` (ASL 2.0), `Bespin` (MPL) и `canvas.text.js` (MIT). Полный список всех правовых положений и замечаний об авторских правах можно найти по адресу: **http://diveintohtml5.org/legal.html**. Всем коллегам, давшим свое разрешение на использование в книге своих разработок и ценные советы, я весьма признателен. По ходу изложения автор приносит благодарности и всем, кто помог ему ценным советом в процессе работы над этой книгой.

Марк Пилгрим

Введение

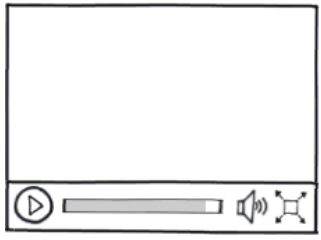
Что представляет собой HTML5? Это — следующее поколение стандартов HTML 4.01[1], XHTML 1.0[2] и XHTML 1.1[3]. HTML5 предоставляет новые функциональные возможности, необходимые современным Web-приложениям. Кроме того, этот стандарт регламентирует множество функций Web-платформы, которые Web-разработчики использовали годами, но которые никогда не проверялись и не документировались комитетом по стандартизации. Скажем, удивит ли вас тот факт, что объект `Window` никогда не был формально документирован? Помимо того что HTML5 вводит новые функции, спецификация HTML5 представляет собой первую попытку формально документировать все стандарты "де-факто", которые Web-браузеры использовали годами.

Пять фактов, которые необходимо знать о HTML5

Прежде чем углубляться в детали, приведем пять ключевых фактов о HTML5, которые должен знать каждый:

1. HTML5 — это не есть некая большая, единая, "монолитная" сущность.

   У вас может возникнуть вопрос: "Как мне начать работать с HTML5, если старые браузеры его не поддерживают?" Но этот вопрос поставлен некорректно, потому что он уводит вас в сторону от обсуждаемой темы. Дело в том, что HTML5 — это не есть нечто



[1] Подробнее см. **http://www.w3.org/TR/html401/**, **http://www.w3.org/TR/html401/struct/global.html**, **http://www.umade.ru/resources/specifications/html401/index.htm**. — *Прим. перев*.

[2] XHTML (Extensible Hypertext Markup Language — Расширяемый язык разметки гипертекста) — язык разметки Web-страниц, по возможностям сопоставимый с HTML, созданный на базе XML. Подробнее см. **http://www.w3.org/TR/xhtml1/**, **http://www.opennet.ru/docs/RUS/XHTML1/**. — *Прим. перев*.

[3] Подробнее см. **http://www.w3.org/TR/xhtml11/**, **http://www.opennet.ru/docs/RUS/XHTML11/**. — *Прим. перев*.

единое и неделимое. Напротив, данная спецификация представляет собой набор индивидуальных функций и возможностей. Поэтому нет никакого смысла пытаться дать общее определение термину "поддержка HTML5". Однако, несмотря на это, вполне возможно дать определения поддержке отдельных функциональных возможностей, например, таких как элементы `<canvas>`, `<video>` или `<geolocation>`.

Возможно, вы думаете о HTML как о наборе тегов и угловых скобок. Это отчасти правильно, потому что теги и угловые скобки действительно представляют собой важную часть HTML. Важную, но не единственную, и это — еще не все. Спецификация HTML5 также определяет, каким образом все эти теги и угловые скобки используют объектную модель документов (Document Object Model, DOM[1]) для взаимодействия с JavaScript. Так, спецификация HTML5 не просто определяет тег `<video>`; в ней определяется и соответствующий интерфейс прикладного программирования DOM (DOM API) для обработки объектов видео в DOM. Вы можете использовать этот API, чтобы обнаруживать поддержку различных форматов видео, воспроизводить видео, прерывать воспроизведение, воспроизводить аудио в немом режиме (mute), отслеживать, какой объем видео был загружен, а также делать все, что вам необходимо для построения пользовательского интерфейса приложений, богатого функциональными возможностями, базирующимися на теге `<video>`.

В *главе 2* и *приложении 1* будет показано, как правильно определять наличие поддержки браузером каждой из новых возможностей HTML5.

2. Вам ничего не придется "выбрасывать".

Нравится вам это или нет, но вы не можете отрицать бесспорного факта: HTML 4 — это самый успешный из всех форматов разметки, существовавших до сих пор. Формат HTML5 строится на базе HTML 4, на волне этого успеха. Вам нет никакой необходимости отказываться от имеющихся у вас "наработок" — вашей уже существующей разметки. Вам не требуется заново учить или переучивать то, что вы уже знаете. Если ваше Web-приложение работает сегодня, используя HTML 4, то оно будет работать и завтра, используя HTML5. На этом можно поставить точку.

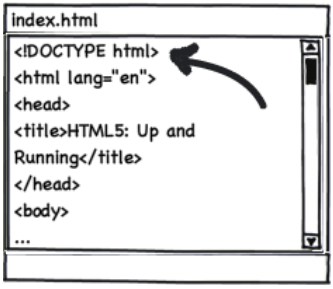
Зато, если вам хочется *усовершенствовать* свои Web-приложения, созданные ранее, то данная книга — это как раз то, что вам и требуется. Вот реальный пример: HTML5 поддерживает все элементы управления формами, которые имелись и в HTML 4, но, кроме того, добавляет и новые элементы, в том числе — новые элементы управления вводом. Некоторые из них представляют со-

[1] DOM (Document Object Model — "объектная модель документа") — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получать доступ к содержимому документов на языках HTML, XHTML и XML, а также изменять содержимое, структуру и оформление таких документов. Подробнее см. **http://www.w3.org/DOM/**, **http://javascript.ru/tutorial/dom**. — *Прим. перев.*

бой долгожданные дополнения, наподобие "ползунковых регуляторов" (sliders) и элементов выбора даты (date pickers); другие же не так просты. Например, элемент ввода данных типа `email` выглядит как обычное текстовое поле, но мобильные браузеры могут настраивать свою экранную клавиатуру так, чтобы пользователю было удобнее вводить почтовые адреса. Более старые браузеры, которые не поддерживают элемента ввода данных типа `email`, будут обращаться с этим полем точно так же, как они обычно обращаются с текстовыми полями, и форма все равно будет работать, без всякой необходимости вносить модификации в ее код или в скрипты. Это означает, что вы уже сегодня можете начать работать над усовершенствованием Web-форм, несмотря даже на то, что некоторые из посетителей вашего сайта могут продолжать пользоваться браузером Internet Explorer 6.

Подробная и детальная информация о формах HTML5 будет приведена в *главе 9*.

3. Начать работу с HTML5 очень просто.

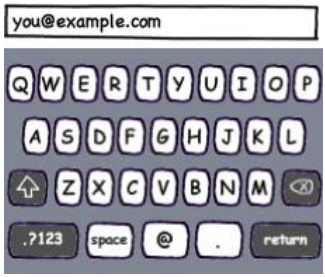


"Обновление" до HTML5 может оказаться не сложнее, чем изменение типа документа (*doctype*). Тип документа должен быть указан уже в первой строке любой страницы HTML. Предшествующие версии HTML определяли большое количество типов документов (doctypes), и выбор правильного типа документа из числа доступных мог представлять проблему. В HTML5, напротив, существует только один тип документа:

```
<!DOCTYPE html>
```

Обновление до типа документа HTML5 не разрушит вашу существующую разметку, потому что теги, определенные в HTML 4, по-прежнему поддерживаются и в HTML5. Но теперь вы сможете определять, использовать и новые семантические элементы наподобие `<article>`, `<section>`, `<header>` и `<footer>`, а также проверять их правильность. Об использовании этих элементов подробно рассказывается в *главе 3*.

4. Все работает уже сейчас.



Не имеет значения, что вы хотите делать с элементом `<canvas>`: рисовать на нем, воспроизводить видео, конструировать новые, усовершенствованные формы или строить Web-приложения, работающие автономно (offline), — вы быстро убедитесь в том, что спецификация HTML5 уже сегодня обеспечивается мощной поддержкой. Большинство популярных Web-браузеров, в том числе — Mozilla Firefox, Safari, Google Chrome, Opera, а также мобильные браузеры уже поддерживают такие элементы новой спецификации, как `<canvas>` (*глава 4*), `<video>` (*глава 5*), `<geolocation>` (*глава 6*), локальное хранилище `<localstorage>` (*глава 7*) и многое другое. Google уже поддерживает аннотации микроданных (microdata

annotations), о чем будет рассказано в *главе 10*. Даже корпорация Microsoft, которая редко бывает первопроходцем в пунктуальном следовании новым стандартам (особенно тем, которые находятся на стадии разработки и еще не утверждены), будет поддерживать большинство функций HTML5 в ожидаемом релизе своего браузера Internet Explorer 9[1].

Каждая глава этой книги содержит диаграммы совместимости для всех широко известных браузеров. Что еще важнее, в каждой главе приводится открытая дискуссия о том, какие опции вам нужны для поддержки старых браузеров. Такие элементы HTML5, как `<geolocation>` (*глава 6*) и `<video>` (*глава 5*) оказались первыми функциями, предоставляемыми плагинами браузеров, наподобие Gears[2] и Flash[3].

Некоторые другие функции, например, `<canvas>` (*глава 4*), могут полностью эмулироваться с помощью JavaScript. В данной книге будет показано, как обращаться к этим функциям, встроенная поддержка которых обеспечивается новейшими версиями браузеров, не отказываясь при этом от старых версий.

5. HTML5 уже здесь, и это — всерьез и надолго.

Как известно всем, Тим Бернерс-Ли (Tim Berners-Lee) изобрел Всемирную паутину (World Wide Web) в начале 1990-х годов. Впоследствии он основал Консорциум Всемирной паутины (World Wide Web Consortium, W3C), призванный координировать работу по созданию стандартов Web, чем эта организация и занималась в течение более чем 15 лет. Вот какое сообщение было сделано W3C относительно будущего Web-стандартов в июле 2009 года:

"На сегодня Директор объявляет о том, что, когда истечет срок действия текущего рабочего соглашения на право ведения деятельности для рабочей группы XHTML 2, намеченный на конец 2009 года, это соглашение продлено не будет. Поступая таким образом и выделяя основные ресурсы рабочей группе HTML, W3C

---

[1] Фактически, к моменту подготовки русского издания книги, этот браузер уже вышел (релиз-кандидат был представлен 10 февраля 2011 года). Подробнее о поддержке HTML5 в Internet Explorer 9 можно прочесть по следующим адресам: **http://www.coolwebmasters.com/browsers/1011-html5-support-internet-explorer-9.html**, **http://habrahabr.ru/blogs/ie/94968/**, **http://internet.cnews.ru/reviews/index.shtml?2010/12/17/420552**, **http://internet.cnews.ru/reviews/index.shtml?2010/12/17/420552**. — *Прим. перев.*

[2] Gears (ранее Google Gears) — открытое ПО от Google, позволяющее использование Web-приложений в режиме оффлайн. Еще 30 ноября 2009 года компания Google объявила: "Мы продолжаем поддержку Gears, так что ничего не сломается на сайтах, которые его используют. Но мы ожидаем, что разработчики будут использовать HTML5 для получения нужных функциональных возможностей, переходя на ориентированный на стандарты подход, который будет применим во всех браузерах". Подробнее см. **http://latimesblogs.latimes.com/technology/2009/11/google-gears.html**. — *Прим. перев.*

[3] Adobe Flash (ранее Macromedia Flash), или просто Flash (по-русски часто пишут флеш или флэш) — мультимедийная платформа компании Adobe для создания Web-приложений или мультимедийных презентаций. Широко используется для создания рекламных баннеров, анимации, игр, а также воспроизведения на Web-страницах видео- и аудиозаписей. Подробнее см. **http://www.adobe.com/ru/products/flash/?promoid=BPCEP**. — *Прим. перев.*

выражает надежду на ускорение развития HTML5 и проясняет свою позицию относительно будущего HTML"[1].

Так что HTML5 — это всерьез и надолго. А теперь давайте приступим к детальному изучению этой новой спецификации.

В данной книге будут рассмотрены следующие основные темы:

- Новые семантические элементы наподобие `<header>`, `<footer>` и `<section>` (*глава 3*);
- Элемент `<canvas>`, двумерная поверхность, содержимое которой можно программировать на JavaScript (*глава 4*);
- Элемент `<video>`, который можно встраивать в Web-страницы без необходимости прибегать к сторонним плагинам (*глава 5*);
- Элемент `<geolocation>`, который посетители вашего Web-сайта смогут выбирать для публикации своих географических координат в вашем Web-приложении;
- Постоянное локальное хранилище, использование которого не будет требовать использования сторонних плагинов (*глава 7*);
- Возможности автономной работы Web-приложений даже в случае разрыва сетевых соединений (*глава 8*);
- Усовершенствованные Web-формы HTML (*глава 9*);
- Микроданные (microdata), которые позволяют вам создавать собственные словари и встраивать семантическую разметку в документы HTML5 (*глава 10*).

Разработка HTML5 ведется таким образом, чтобы, по мере возможности, обеспечивать обратную совместимость с существующими Web-браузерами. Новые функции строятся на основе уже существующих и позволяют вам создавать информационное содержимое, которое (с игнорированием функций HTML5) может отображаться и более старыми версиями браузеров. Если вам требуется еще более высокий уровень контроля, вы можете выявлять поддержку браузером отдельных функций HTML5, используя всего лишь несколько строк на языке JavaScript. Не следует полагаться в этом на слабые возможности самих браузеров. Вместо этого вы можете протестировать поддержку нужной вам функции HTML5 самостоятельно. О том, как это делается, рассказывается в *главе 2*.

Соглашения, используемые в данной книге

В оформлении этой книги используются следующие соглашения о типографском оформлении:

- Текстом *курсивного начертания* выделяются новые термины;
- Текстом **полужирного начертания** выделяются почтовые адреса, URL и интерфейсные элементы;
- `Моноширинный шрифт` используется для листингов программ, имен переменных и функций, баз данных, типов данных, переменных окружения, утверждений и ключевых слов, имен файлов и расширений имен файлов;

[1] См. **http://www.w3.org/MarkUp/**. — *Прим. перев.*

- **`Моноширинный шрифт полужирного начертания`** используется для выделения команд или иного текста, который пользователь должен вводить вручную;
- *`Моноширинный шрифт курсивного начертания`* используется для выделения значений, которые должны быть заменены на пользовательский ввод, а также для контекстно-зависимых значений;

Использование примеров кода

Эта книга написана с тем, чтобы помочь вам справляться с вашей работой. В общем случае, вы можете использовать код, приведенный в книге, в ваших программах и написанной вами документации. Вам не требуется обращаться к нам за разрешением на использование материалов, за тем исключением, когда вы заимствуете значительные фрагменты кода. К примеру, написание программы с использованием нескольких небольших фрагментов кода из этой книги не требует обращения за разрешением. Но тиражирование и продажа носителей CD-ROM с примерами из этой или других книг O'Reilly разрешения требует. Если вы отвечаете на чей-то вопрос и приводите цитату со ссылкой на данную книгу, то такое использование разрешения не требует. Но для включения масштабного фрагмента, заимствованного из книги O'Reilly, в документацию к вашему продукту разрешение требуется.

Мы приветствуем библиографические ссылки, хотя и не настаиваем на точном соблюдении формы, в которой они приводятся. Такие ссылки обычно включают заголовок, имя автора, издательство и ISBN (International Standard Book Number, стандартный международный номер книги). Например: "*HTML5: Up and Running*, by Mark Pilgrim. Copyright 2010 O'Reilly Media, Inc., 978-0-596-80606-6."

Если вам кажется, что использование кода примеров выходит за рамки законного применения или только что приведенных условий получения разрешения, вы можете связаться с нами по адресу **permissions@oreilly.com**.

Safari® Books Online

Safari Books Online — это электронная библиотека "по требованию", позволяющая легко найти более 7 500 оригинальных справочных пособий и видео о передовых технологиях для получения быстрых ответов на интересующие вас вопросы.

Оформив подписку на сервис Safari Books Online, вы сможете прочесть любую страницу и просмотреть любой видеофильм из нашей интерактивной библиотеки. Читать книги можно на мобильном телефоне или других мобильных устройствах. Вы сможете просматривать заголовки новых книг до того, как они выйдут из печати, получать эксклюзивный доступ к рукописям, готовящимся к печати, а также возможность связаться по почте с их авторами. Более того, вы сможете копировать и вставлять код примеров, формировать свои списки "Избранного", скачивать отдельные главы, создавать закладки в интересующих вас разделах, создавать примечания, печатать страницы и пользоваться множеством экономящих время функций.

Издательство O'Reilly Media включило данную книгу в библиотеку Safari Books Online. Для получения полного доступа к этой книге и другим книгам сходной тематики издательства O'Reilly и других издательств подключайтесь бесплатно на Web-сайте **http://my.safaribooksonline.com**.

Как связаться с издательством O'Reilly

Пожалуйста, посылайте замечания и вопросы, относящиеся к этой книге, издателю:

O'Reilly Media Inc.,
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США и Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

У нас есть посвященная этой книге Web-страница, на которой приведен список опечаток, примеры и другая дополнительная информация. К этой странице можно обратиться по адресу:

http://oreilly.com/catalog/9780596806066

У этой книги есть и свой Web-сайт по адресу:

http://diveintohtml5.org/

Для отправки замечаний и вопросов, касающихся этой книги, отправляйте письма с указанием ISBN-номера (**9780596806066**) по адресу:

bookquestions@oreilly.com

Для получения дополнительной информации о книгах, конференциях, центрах распространения (Resource Centers) и сервисе O'Reilly Network посетите Web-сайт издательства:

http://www.oreilly.com

Замечания об изданиях этой книги

Эта книга создана на основе исходного текста HTML5, размещенного по адресу **http://diveintohtml5.org/** и поддерживаемого автором. Электронные версии книги (**http://oreilly.com/catalog/9780596806066**) в форматах ePub, Mobi и PDF, свободных от ограничений DRM, как и версия Safari Books Online (**http://my.safaribooksonline.com/9781449392154**), содержат все гиперссылки, присутствующие в оригинале, в то время как в печатное издание включено лишь подмножество гиперссылок. Если вы читаете печатное издание, обращайтесь к одному из электронных изданий или к исходному варианту, где вы найдете гораздо большее количество ссылок. Поскольку автор поддерживает сайт **http://diveintohtml5.org/** в HTML5, там вы найдете живые примеры кода, описанные в этой книге, многие из которых при подготовке печатного издания были модифицированы. При просмотре примеров на сайте **http://diveintohtml5.org/** следует иметь в виду, что их визуализация зависит от используемого вами браузера.

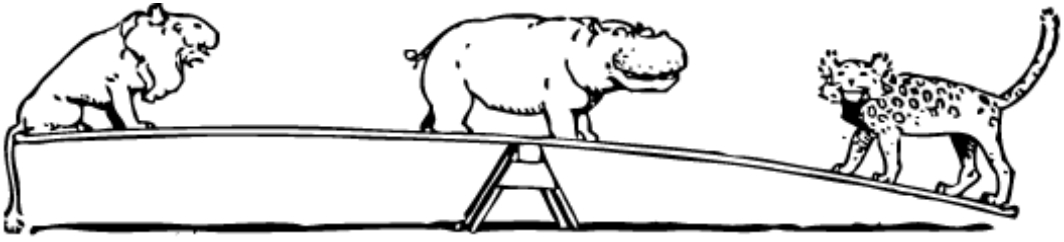
Глава 1

Как мы пришли туда, где находимся?

Недавно я натолкнулся на цитату, взятую из текста, написанного одним из разработчиков Mozilla о трениях, которые внутренне присущи командам, занятым разработкой стандартов (**http://lists.w3.org/Archives/Public/public-html/2010Jan/0107.html**):

"Реализации и спецификации должны взаимодействовать между собой крайне осторожно. Никто не хочет, чтобы реализации появлялись до того, как будет завершена работа над спецификацией, потому что тогда пользователи будут зависеть и от деталей реализации, и от ограничений спецификации. Но при этом никто не хочет и такого развития событий, когда спецификация оказывается уже законченной, принятой и утвержденной, но ни один из авторов и разработчиков ПО еще не получил никакого опыта реализации проектов, основанных на этой спецификации. Дело в том, что разработчикам спецификаций тоже нужна обратная связь. Это — неизбежный источник напряженности и трений, но нам просто необходимо преодолеть и эту напряженность, и эти трения".

Запомните эту цитату, зарубите ее себе на носу, запишите ее в свой блокнот. А теперь давайте разбираться с тем, что представляет собой стандарт HTML5 и что вызвало его к жизни.



Типы MIME

Эта книга посвящена HTML5, а не предшествующим версиям HTML, и не более ранним версиям XHTML. Но для понимания истории появления и развития HTML5, как и мотивов, послуживших движущими силами развития этого стандар-

та, вам необходимо сначала разобраться с некоторыми техническими деталями, с частности, с *типами MIME*[1].

Каждый раз, когда ваш Web-браузер запрашивает страницу, Web-сервер сначала отсылает ему *заголовки* (headers), и только затем — фактическую разметку страницы (page markup). Обычно эти заголовки невидимы, несмотря на то, что на самом деле они представляют собой средства Web-разработки, которые станут видимыми, если вы в этом заинтересованы. Заголовки играют важную роль, потому что они сообщают вашему браузеру, как следует интерпретировать разметку страницы, которая будет получена браузером впоследствии. Наиболее важный заголовок называется `Content-Type` и выглядит он так:

```
Content-Type: text/html
```

Строка `text/html` называется "типом содержимого" (или "типом контента", или "MIME-типом") страницы. Этот заголовок единолично определяет тип ресурса, который представляет собой страница, и, следовательно, указывает, как эта страница должна быть визуализирована. Изображения имеют собственные типы MIME (например, `image/jpeg` — для изображений формата JPEG, `image/png` — для изображений формата PNG и т. д.). Файлы JavaScript имеют собственный тип MIME. Каскадные страницы стилей (CSS) тоже имеют свой собственный тип MIME. Все имеет свой тип MIME. Вся Всемирная Паутина работает благодаря типам MIME.

Естественно, окружающая нас реальность гораздо сложнее. Первое поколение Web-серверов (а я рассматриваю Web-серверы, появившиеся, начиная с 1993 года) не отправляло заголовков `Content-Type`, потому что таких заголовков в те времена еще не существовало (они были изобретены после 1994 года). В целях обеспечения обратной совместимости с продуктами, датируемыми вплоть до 1993 года, некоторые популярные браузеры при определенных обстоятельствах игнорируют заголовок `Content-Type`. Этот прием называется "сниффингом контента" (content sniffing) или "распознаванием браузера" (browser sniffing). Но, как общее эмпирическое правило, все, что вы когда-либо просматривали через Web, включая страницы HTML, изображения, скрипты, видео, PDF-файлы, и все, что имеет URL, предоставлялось вам с указанием конкретного типа MIME в заголовке `Content-Type`.

Запомните этот факт. Впоследствии мы к этому еще вернемся.

Длинный экскурс в историю разработки стандартов

Почему существует элемент `<img>`? Естественно, это не тот вопрос, который вы слышите каждый день. Очевидно, что *кто-то* его создал. Такие вещи не появ-

[1] Multipurpose Internet Mail Extensions (MIME) — многоцелевые расширения почты Интернета, произносится как "майм"). Это стандарт, описывающий передачу различных типов данных по электронной почте, а также, в более широком понимании — спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету. Подробнее см. **http://ru.wikipedia.org/wiki/MIME**, **http://www.relcom.ru/Internet/Services/Email/MIME/**, **http://www.mhonarc.org/~ehood/MIME/**. — *Прим. перев.*

ляются из ниоткуда. Каждый элемент, каждый атрибут, каждая функция HTML, которой вы когда-либо пользовались, определенно, кем-то были созданы. Тот, кто их создал, продумал, как они должны работать, и написал все это. Все эти люди — не боги, и они далеко не безупречны. Они просто люди. Умные люди, конечно, но просто люди.

Одной из замечательных особенностей стандартов, которые разрабатываются открыто и сразу же становятся достоянием общественности, является то, что вы можете совершить экскурс в прошлое и ответить на некоторые из этих вопросов. Дискуссии происходят в списках рассылки, которые обычно архивируются и доступны для публичного поиска. Поэтому я решил немного заняться "почтовыми раскопками", чтобы попытаться ответить на вопрос о том, откуда взялся элемент `<img>`. Мне пришлось сделать экскурс в те времена, когда еще не существовало такой организации, как World Wide Web Consortium (W3C). Я вернулся к самым ранним дням существования Web, когда количество Web-серверов по всему миру можно было сосчитать по пальцам рук (ну, возможно, задействовав дополнительно еще парочку пальцев и на ногах).

25 февраля 1993 года Марк Андреессен (Mark Andreessen) написал (**http://1997.webhistory.org/www.lists/www-talk.1993q1/0182.html**)[1]:

"Я хотел бы предложить новый, необязательный тег HTML:

```
IMG
```

Обязательный аргумент выглядит так: `SRC="url"`.

Этот тег указывает браузеру растровый или пиксельный файл, предназначенные для извлечения через локальную сеть или Интернет в виде изображения, которое должно быть встроено в текст в той точке, где указан тег.

Рассмотрим пример:

```
<IMG SRC="file://foobar.com/foo/bar/blargh.xbm">
```

(Закрывающего тега нет, это просто самостоятельный тег.)

Этот тег может встраиваться в качестве якоря как и любой другой; когда это происходит, он становится значком, чувствительным к активации, как и любой текстовый якорь.

Браузерам следует дать возможность выбирать, какие форматы изображений они будут поддерживать. Например, Xbm и Xpm вполне подходят для поддержки. Если браузер не может интерпретировать данный формат, он вместо отображения может выбрать любое действие в соответствии со своими предпочтениями (например, X Mosaic выведет в качестве метки-заполнителя стандартное растровое изображение).

[1] Ветвь дискуссии, обсуждаемую на протяжении последующих нескольких страниц, можно отслеживать, выполняя щелчки мышью по ссылкам **Next message** и **Previous message**.

Это — необходимая функциональность для X Mosaic; мы добились того, что это работает, и теперь собираемся применять эту функциональность хотя бы для внутреннего пользования. Я открыт для предложений по использованию этой функции в HTML; если у вас есть лучшие идеи, чем та, которую я представил здесь, сообщите мне об этом, пожалуйста. Я знаю, что здесь есть неопределенность относительно формата изображения, но я пока не вижу никакого альтернативного варианта, отличного от того, чтобы просто сказать "дайте браузеру возможность сделать то, что он может" и подождать, пока со временем не появится лучшее решение (возможно, со временем, MIME)".

Приведенная цитата нуждается в некоторых пояснениях:

Xbm[1] и Xpm[2] представляли собой популярные графические форматы в системах UNIX.

"Mosaic" представлял собой один из самых ранних Web-браузеров, а "X Mosaic" — версию, которая работала в UNIX-системах. На момент написания этого сообщения в начале 1993 года, Марк Андреессен (Mark Andreessen) еще не основал компанию, которая сделала его знаменитым, Mosaic Communications Corporation, и еще не начинал работать над продуктом, который стал "флагманом" этой компании, "Mosaic Netscape" (хотя возможно, вы лучше знаете и компанию, и ее продукт по именам, которые они получили впоследствии: "Netscape Corporation" и "Netscape Navigator").

Упоминание "возможно, со временем, MIME" представляет собой ссылку на согласование содержимого (content negotiation), функцию HTTP, при которой клиент (например, Web-браузер) сообщает серверу (например, Web-серверу) о том, какие типы ресурсов он поддерживает (например, `image/jpeg`), чтобы сервер мог возвращать данные в формате, предпочитаемом клиентом. Оригинальный вариант HTTP был определен в 1991 году, и единственная его реализация, существовавшая на февраль 1993, не предоставляла клиентам способа сообщить серверу о том, какие типы изображений они поддерживают, отсюда и дилемма, с которой столкнулся Марк.

Через несколько часов был получен ответ от Тони Джонсона (Tony Johnson):

"У меня в Midas 2.0 тоже есть очень похожая вещь (и мы пользуемся ею в SLAC и выпустим ее в ближайшем публичном релизе, который должен произойти с недели на неделю), за тем исключением, что все имена отличаются, и функция имеет дополнительный аргумент `NAME="name"`. Возможности функции примерно такие же, как все то, что обеспечивается предложенным вами тегом `IMG`, например:

```
<ICON name="NoEntry" href="http://note/foo/bar/NoEntry.xbm">
```

[1] В компьютерной графике, текстовый формат для монохромных изображений, применяется в X Windows System для хранения изображений курсоров и значков (icons), используемых в X GUI. Подробнее см. **http://en.wikipedia.org/wiki/X_BitMap**. — *Прим. перев.*

[2] X Pixmap (XPM) — текстовый формат графических файлов. Используется серверной частью системы X Windows. Главной особенностью формата является то, что он использует семантику языка C (Си) и может быть включен в код. Подробнее см. **http://en.wikipedia.org/wiki/X_PixMap**, **http://openfile.ru/types/xpm**. — *Прим. перев.*

Смысл параметра `name` в том, чтобы дать браузеру возможность иметь набор "встроенных" изображений. Если значение `name` совпадет с именем "встроенного" изображения, браузер сможет использовать его вместо того, чтобы искать и извлекать изображение. Кроме того, имя может служить и как подсказка для "линейных" браузеров, сообщающая им, какой символ нужно подставить вместо изображения.

Я не придаю слишком важного значения именам параметров и тегов, но считаю, что было бы благоразумно, если бы мы использовали одно и то же. Я не слишком озабочен сокращениями, так что почему бы не использовать `IMAGE=` и `SOURCE=`. Я склоняюсь к слову `ICON`, потому что оно подразумевает, что изображение (`IMAGE`) должно быть небольшим, но не будет ли `ICON` слишком "тяжеловесным" словом?"

Midas являлся еще одним ранним Web-браузером, "современником" X Mosaic. Это был кросс-платформенный браузер; он работал как на UNIX, так и на VMS[1]. Аббревиатура SLAC означала "Стэнфордский центр линейного ускорителя" (Stanford Linear Accelerator Center), ныне — Национальная ускорительная лаборатория SLAC (SLAC National Accelerator Laboratory), исследовательскую лабораторию, в которой располагался первый Web-сервер в США (и первый Web-сервер за пределами Европы[2]). На тот момент, когда Тони (**http://www.slac.stanford.edu/history/earlyweb/wizards.shtml#Tony%20Johnson**) написал это сообщение, SLAC уже был "ветераном" WWW, который содержал на своем сервере целых пять Web-страниц в течение 441 дня (и на то время этот срок был гигантским).

Затем Тони продолжил:

"Хотя мы обсуждаем новые теги, у меня есть еще один, в чем-то похожий тег, который я хотел бы поддерживать в Midas 2.0. В принципе, он выглядит так:

```
<INCLUDE HREF="...">
```

Здесь мои намерения заключаются в том, что я хотел бы, чтобы в первый документ был включен второй, как раз в месте вхождения тега. В принципе, документ, на который дается ссылка, может представлять собой что угодно, но основной целью является обеспечение возможности встраивать в документы изображения (в данном случае размер изображения может быть произвольным). Опять же, намерения таковы, чтобы после появления HTTP2 формат включенного документа стал предметом для отдельного согласования".

Здесь "HTTP2" относится к протоколу Basic HTTP в том виде, в котором он был определен в 1992 году. На тот момент, в начале 1993 года, он в значительной степени был еще нереализованным. Черновик стандарта, на тот момент известного

---

[1] Серверная ОС, разработанная во второй половине 1970-х компанией Digital Equipment Corporation для компьютеров VAX (см. **http://ru.wikipedia.org/wiki/VAX**). Впоследствии была портирована на платформы DEC Alpha и Intel Itanium. Сейчас принадлежит Hewlett-Packard. Подробнее см. **http://wikiadmin.net/OpenVMS**, **http://h71000.www7.hp.com/index.html?jumpid=/go/openvms**. — *Прим. перев.*

[2] См. **http://www.slac.stanford.edu/history/earlyweb/history.shtml**, **http://www.slac.stanford.edu/history/earlyweb/firstpages.shtml**. — *Прим. перев.*

как "HTTP2", продолжал развиваться, и со временем был утвержден как стандарт "HTTP 1.0" (хоть и произошло это через целых три года). Стандарт HTTP 1.0[1] включал заголовки для согласования контента (**http://www.w3.org/Protocols/HTTP/HTRQ_Headers.html#z3**), иначе говоря, так было реализовано предположение "возможно, со временем, MIME".

Далее Тони продолжил развивать свою идею:

"Альтернатива, которую я рассматривал, выглядит так:

```
<A HREF="..." INCLUDE>See photo</A>
```

Я не слишком стремлюсь добавлять функциональные возможности тегу `<A>`, но основная идея заключается в том, чтобы поддерживать совместимость с браузерами, которые не принимают во внимание параметр `INCLUDE`. Мое намерение заключается в том, чтобы браузеры, которые интерпретируют `INCLUDE`, замещали текст привязки (в данном случае "`See photo`") встраиваемым документом (картинкой), а старые или менее продвинутые браузеры полностью игнорировали тег `INCLUDE`".

Это предложение никогда не было воплощено в жизнь, хотя сама идея предоставления текста в случае отсутствия изображения представляла собой важный прием (**http://diveintoaccessibility.org/day_23_providing_text_equivalents_for_images.html**), которого не было в изначальном предложении тега `<IMG>`, сделанного Марком. Год спустя эта функция была искусственно "прикручена" как атрибут `<img alt>`, который Netscape сразу же "сломал" (**http://www.cs.tut.fi/~jkorpela/html/alt.html#tooltip**), по ошибке интерпретируя его как всплывающую подсказку (tooltip).

Через несколько часов после того, как Тони отправил свое сообщение, на это сообщение ответил Тим Бернерс-Ли (Tim Berners-Lee):

Я предполагал следующее представление:

```
<a name=fig1 href="fghjkdfghj" REL="EMBED, PRESENT">Figure </a>
```

где соотношение между значениями обозначало бы следующее:

`EMBED`	Встроить сюда при представлении
`PRESENT`	Отобразить, если представлен исходный документ

Обратите внимание, что вы можете использовать различные комбинации, и, если браузер не поддерживает ни одной из этих комбинаций, он не "падает".

[Я] вижу, что использование этого метода для выбираемых значков означает вставку "якорей" (anchors). Хм. Но я не хотел вводить специальный тег".

Это предложение принято не было, но атрибут `rel` по-прежнему существует.

Затем Джим Дэвис (Jim Davis) добавил:

"Было бы неплохо, если бы существовал способ указывать тип содержимого, например:

```
<IMG HREF="http://nsa.gov/pub/sounds/gorby.au" CONTENT-TYPE=
audio/basic>
```

Но меня вполне удовлетворит и жизнь с требованием того, чтобы я самостоятельно указывал тип по расширению имени файла".

---

[1] См. **http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html**, **http://www8.org/w8-papers/5c-protocols/key/key.html**. — *Прим. перев.*

Это предложение тоже не было реализовано, но впоследствии Netscape добавил поддержку для мультимедийных объектов с помощью элемента `<embed>`.

Джей Си Вебер (Jay C. Weber) спросил:

"Хотя изображения имеют приоритет в моем списке пожеланий предпочитаемых типов информации для WWW-браузера, я не думаю, что мы должны добавлять специфические обработчики для характерных типов информации по одному за раз. Куда подевался весь энтузиазм по поводу использования механизма типов MIME?"

Марк Андреессен ответил на это сообщение так:

"Это не замена ожидаемому применению MIME как стандартного механизма для работы с документами; это всего лишь простая и необходимая реализация функциональной возможности, которая нужна независимо от MIME".

Джей Си Вебер (Jay C. Weber) на это отвечал:

"Давайте на время забудем о MIME, если это затемняет суть вопроса. Я возразил против того, чтобы начинать дискуссию о том, "как мы собираемся поддерживать встроенные изображения", а не о том, "как мы собираемся поддерживать встроенные изображения в различных средах распространения информации".

В противном случае, на следующей неделе кто-нибудь предложит новый тег "`<AUD SRC="file://foobar.com/foo/bar/blargh.snd">`" для звуковой информации.

Не нужно тратить слишком много усилий на то, что можно обобщить".

С учетом ретроспективного взгляда, складывается впечатление, что доводы Джея были хорошо обоснованы. Конечно, времени понадобилось больше недели, но, в конце концов, в HTML5 элементы `<video>` и `<audio>` действительно были все-таки добавлены.

Отвечая на изначальное сообщение Джея, Дэйв Рэггетт (Dave Raggett) заметил:

"Естественно, вы правы! Я хотел рассмотреть весь диапазон всевозможных типов изображений/штриховой графики, вместе с возможностью согласования форматов. Важное значение имеет и замечание Тима о возможности поддержки областей с возможностью щелчков мышью в пределах изображений".

Впоследствии, в 1993 году, Дэйв Рэггетт (Dave Raggett) предложил HTML+ (**http://www.w3.org/MarkUp/HTMLPlus/htmlplus_1.html**) как дальнейшее развитие стандарта HTML. Это предложение реализовано не было, и впоследствии было заменено спецификацией HTML 2.0 (**http://www.w3.org/MarkUp/html-spec/html-spec_toc.html**). HTML 2.0 представляла собой "ретроспецификацию", в том смысле, что она просто формализовала те функции, которые использовались уже повсеместно **(http://www.w3.org/MarkUp/html-spec/html-spec_1.html#SEC1.1**). Эта спецификация сводила воедино, проясняла и формализовала набор функциональных возможностей, который приблизительно соответствовал возможностям спецификации HTML, в том виде, в котором она широко использовалась до начала июня 1994 года.

Впоследствии Дэйв написал спецификацию HTML 3.0 **(http://www.w3.org/MarkUp/html3/CoverPage.html**), которая основывалась на его раннем черновике

HTML+. За пределами собственной реализации W3C, Arena (**http://www.w3.org/Arena/**), спецификация HTML 3.0 так и не была никогда воплощена в жизнь, а затем была вытеснена HTML 3.2 (**http://www.w3.org/MarkUp/Wilbur/**), еще одной "ретро" спецификацией. Спецификация HTML 3.2 добавляла такие широко распространенные функции, как таблицы (tables), апплеты (applets) и обтекание текста вокруг изображений, при одновременном обеспечении обратной совместимости с существующим стандартом HTML 2.0 (**http://www.w3.org/TR/REC-html32.html#intro**).

Впоследствии Дэйв выступил одним из соавторов HTML 4.0 (**http://www.w3.org/TR/html4/**), разработал HTML Tidy (**http://tidy.sourceforge.net/**), а также принял участие в работе над XHTML, XForms, MathML и множеством других современных спецификаций W3C.

Если вернуться к нашей ретроспективе, то в 1993 Марк ответил Дэйву:

"На самом деле, возможно, нам следовало бы задуматься об общецелевом процедурном языке обработки графики, с помощью которого мы могли бы встраивать произвольные гиперссылки, указывающие на значки (icons), изображения (images), текст или что угодно другое. Кто-нибудь видел возможности Intermedia, демонстрируемые данным проектом в этом отношении?"

Intermedia (**http://en.wikipedia.org/wiki/Intermedia_%28hypertext%29**) представляла собой гипертекстовый проект, разработанный в Брауновском университете (**http://tinyurl.com/2g89g6q**). Разработка проекта велась с 1985 по 1991 г., приложение работало на A/UX (**http://en.wikipedia.org/wiki/A/UX**, **http://ru.wikipedia.org/wiki/A/UX**)[1], UNIX-подобной операционной системе, предназначенной для ранних компьютеров Macintosh.

Идея "процедурного языка обработки графики общего назначения" со временем набрала популярность. Современные браузеры поддерживают как SVG (Scalable Vector Graphics, декларативный язык разметки со встроенными возможностями поддержки скриптов, см. **http://www.w3.org/Graphics/SVG/**), так и элемент `<canvas>` (процедурно-ориентированный графический API прямого режима, см. **http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-element**), хотя последний начал свое существование как патентованное расширение (см. **http://ln.hixie.ch/?start=1089635050&count=1**), и только потом был включен в "ретроспецификацию" рабочей группой WHATWG (**http://www.whatwg.org/**).

На это замечание ответил Билл Янссен (Bill Janssen):

"Другие системы, на которые следовало бы посмотреть и которые имеют эту ценную возможность, это Andrew и Slate. Система Andrew построена на основе типов `_insets_`, каждый из которых имеет некоторый интересный тип, например, текст (text), растровое изображение (bitmap), рисунок (drawing), анимация (animation), сообщение (message), электронная таблица (spreadsheet) и т. д. Присутствует понятие произвольного рекурсивного встраивания, так что встроить

[1] Интересующимся читателям можно дополнительно порекомендовать следующие ссылки: **http://www.aux-penelope.com/**, **http://www.floodgap.com/retrotech/os/aux/**, **http://christtrek.dyndns.org:8000/doc/aux/faq.html**. — *Прим. перев.*

в документ можно вставку любого другого вида, который поддерживает встраивание. Например, вставку можно добавить в любом месте текста или текстового виджета (widget), или в любую прямоугольную область графического виджета, или же в ячейку электронной таблицы".

В данном случае под "Andrew" понимается система пользовательского интерфейса Andrew (Andrew User Interface System, **http://www-2.cs.cmu.edu/~AUIS/**), хотя в те времена она была широко известна под названием "проект Эндрю" (Andrew Project, см. **http://en.wikipedia.org/wiki/Andrew_Project**).

Тем временем, Томас Файн (Thomas Fine) высказал другую идею:

"Вот мое мнение. Наилучшим способом встраивания изображений в документы WWW является использование MIME. Я уверен, что PostScript уже является одним из поддерживаемых типов MIME, и этот формат очень хорошо справляется со смешанной обработкой текста и графики.

Но вы возражаете, что он не кликабелен? Да, вы правы. Я подозреваю, что ответ на этот вопрос уже дан Display PostScript. Даже если дополнение к стандартному PostScript и не тривиально. Определите "якорную" команду, которая указывала бы URL и использовала текущий путь как замкнутый регион для кнопки. Поскольку postscript отлично работает с путями, это делает тривиальной задачу создания кнопок произвольной формы".

Display PostScript (**http://en.wikipedia.org/wiki/Display_PostScript**) представлял собой экранную технологию визуализации, совместно разработанную Adobe и NeXT (**http://en.wikipedia.org/wiki/NeXT**, **http://ru.wikipedia.org/wiki/NeXT**).

Это предложение никогда не было реализовано, но идея о том, что лучший способ исправить проблемы с HTML заключается в том, чтобы полностью заменить его чем-то другим, по-прежнему время от времени всплывает (см., например: **http://dbaron.org/log/20090707-ex-html**).

Второго марта 1993 года Тим Бернерс-Ли выступил со следующим комментарием:

"HTTP2 позволяет документу содержать любой тип, о котором пользователь заявляет, что его можно обрабатывать, а не только зарегистрированные типы MIME. Поэтому можно экспериментировать. Да, я считаю, что есть повод рассмотреть PostScript с гипертекстом. Я не знаю, достаточно ли возможностей имеется у Display PostScript. Я знаю, что в Adobe пытаются создать собственный "PDF" на базе PostScript, который будет иметь ссылки и будет читаться их собственными патентованными средствами просмотра.

Я думаю, что общий язык вышележащего уровня (может быть, основанный на HyTime?) мог бы помочь стандартам гипертекста и графики/видео развиваться отдельно, и это помогло бы и тем, и другим.

Пусть тег `IMG` будет играть роль `INCLUDE`, и пусть он ссылается на документ произвольного типа. Или пусть это будет `EMBED`, если `INCLUDE` звучит слишком похоже на директиву `include` из cpp, вследствие чего у людей будет складываться впечатление, что исходный код SGML будет разбираться встраиваемо (inline) — так как это не то, что подразумевается разработчиками".

HyTime (**http://www.hytime.org/**) представлял собой раннюю гипертекстовую систему документов, основанную на базе SGML (Standard Generalized Markup

Language, см. **http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language**, **http://www.w3.org/MarkUp/SGML/**). Она часто маячила в ранних дискуссиях, посвященных HTML и, впоследствии — XML.

Предложение Тима о теге `<INCLUDE>` никогда не было реализовано, хотя его отголоски можно обнаружить в элементах `<object>`, `<embed>` и `<iframe>`.

Наконец, 12 марта 1993 года Марк Андреессен вновь вернулся к этой ветви дискуссии:

> "Возвращаясь к обсуждению встраиваемого изображения — я склоняюсь к выпуску Mosaic v. 0.10, где будут поддерживаться инлайновые изображения/растры в форматах GIF и XBM, как отмечалось ранее.
>
> […]
>
> На данном этапе мы не готовы поддерживать `INCLUDE`/`EMBED`. … Поэтому, вероятно, мы удовлетворимся `<IMG SRC="url">` (не `ICON`, потому что не все встраиваемые изображения можно назвать "значками"). На текущий момент, встроенные изображения не будут иметь явных типов содержимого; впоследствии мы планируем обеспечить такую поддержку (хотя и с общей адаптацией MIME). Фактически, процедуры чтения изображений, которые мы используем сейчас, определяют формат изображения "на лету", так что даже расширение имени файла не будет иметь значения".

Непрерывная линия

Я не перестаю восхищаться всеми аспектами данной дискуссии, продолжающейся уже более 17 лет и приведшей к созданию элемента HTML, который применяется практически на каждой Web-странице, которая когда-либо была опубликована. Вот посмотрите:

- HTTP до сих пор существует. HTTP успешно развивался с версии 0.9 до версии 1.0 и, наконец, до версии 1.1. И в настоящее время он все еще продолжает развиваться[1].
- HTML до сих пор существует. Этот рудиментарный формат данных, который изначально даже не поддерживал встроенных изображений, успешно развивался, со временем вышли его версии 2.0, 3.2, 4.0. HTML развивается по непрерывной линии. Да, это запутанная и извилистая линия, но она непрерывна. Существует множество ответвлений от этой общей линии, множество "тупиковых" ветвей эволюционного дерева, множество направлений, которые люди, поставившие себе целью разработку стандартов, обошли, опережая сами себя (а также авторов реализаций). И тем не менее, линия развития HTML не пресекалась никогда. Эта книга была написана в 2010 году, но современные браузеры

[1] См. **http://datatracker.ietf.org/wg/httpbis/charter/**.

по-прежнему отображают Web-страницы, датирующиеся 1990 годом[1]! Например, я только что загрузил одну из таких страниц на свой мобильный телефон Android, и мне даже не было выведено сообщения, предлагающего подождать, пока завершится обработка старого формата…

- Язык HTML всегда развивался в ходе дискуссии между разработчиками браузеров, авторами Web-страниц, разработчиками стандартов и другими людьми, которые сначала заинтересовались этим "разговором об угловых скобках", а потом и включились в него сами. Большинство успешных версий HTML представляют собой "ретроспецификации", стремящиеся одновременно актуализировать все то, что к моменту их выхода стало фактическим стандартом, и одновременно задать правильное направление дальнейшего развития. Любой, кто скажет вам о том, что необходимо "блюсти чистоту" HTML (предположительно, игнорируя интересы разработчиков браузеров, авторов Web-страниц или и тех, и других), на самом деле просто недостаточно хорошо или даже неверно информирован. HTML никогда не был "чистым", и все попытки его "очистить" неизменно заканчивались грандиозными провалами, масштабы которых сравнимы только с масштабами фиаско, которое неизменно терпели все попытки вообще заменить HTML чем-нибудь другим.
- Ни один из браузеров, появившихся с момента 1993 года, не существует в своем первозданном виде. Работа над проектом Netscape Navigator была прекращена в 1998 году (**http://en.wikipedia.org/wiki/History_of_Mozilla_Application_Suite#Open_sourcing_of_Communicator**), после чего исходный код этого браузера был полностью переработан и фактически "переписан с нуля", в результате чего появился новый проект — Mozilla Suite, от которого впоследствии отпочковался проект Firefox (**http://en.wikipedia.org/wiki/History_of_Mozilla_Firefox)**. Internet Explorer начинал свой жизненный путь достаточно скромно под названием "Microsoft Plus! for Windows 95", где он был упакован в единый пакет с рядом тем оформления рабочего стола и игрой Pinball. Но естественно, что развитие этого браузера тоже можно проследить вплоть до самых истоков (см., например: **http://en.wikipedia.org/wiki/Spyglass_Mosaic**).
- Некоторые операционные системы, датируемые 1993 годом, все еще существуют, но уже ни одна из них не идет в ногу со временем с современной WWW. Большинство пользователей современной "всемирной паутины" подключаются к Интернету с современных ПК, работающих под управлением Windows 2000 или более новой версии Windows, с Macintosh, работающего под управлением Mac OS X, или с ПК, на котором установлен один из современных дистрибутивов Linux, или даже с наладонного устройства наподобие iPhone. В 1993 году доминирующей версией Windows была Windows 3.1 (которая тогда конкурировала с OS/2), компьютеры Macintosh работали под управлением System 7, а дистрибутивы Linux распространялись через Usenet. Хотите получить об этом представление? Найдите какого-нибудь убеленного сединами ветерана и заведите с ним разговор о "Trumpet Winsock" (**http://en.wikipedia.org/wiki/Winsock**,

[1] См. **http://www.w3.org/People/Berners-Lee/FAQ.html#Examples**.

http://en.wikipedia.org/wiki/Berkeley_sockets, **http://www.trumpet.com.au/index.php/downloads.html**) или "MacPPP" (**http://www.index-site.com/macppp.html**).

- Некоторые или даже *одни и те же* люди до сих пор занимаются тем, что мы сейчас называем "стандартами Web". И это после 20 лет развития! Причем некоторые из них принимали участие в разработке предшественников HTML, датируемых 1980 годом, и даже более ранними датами.
- Если же говорить о предшественниках… С учетом роста популярности HTML и Web, несложно просто забыть о тех форматах и системах, которые стояли у истоков. Andrew? Intermedia? HyTime? Между прочим, HyTime был стандартом ISO (International Organization for Standardization, **http://www.iso.org/iso/home.html**), а не просто одним из экспериментальных проектов, вышедших из недр исследовательской лаборатории (см. **http://xml.coverpages.org/hytime.html**). Этот стандарт был одобрен для применения в военных целях. Его разработка спонсировалась Большим Бизнесом. И прочесть об этом вы можете сами, вот здесь: **http://www.sgmlsource.com/history/hthist.htm**. Между прочим, страница превосходно откроется любым современным браузером!

Но ничто из только что сказанного не дает ответа на исходный вопрос, которым мы задались в начале этой главы: почему мы пользуемся элементом `<img>`? Почему не элементом `<icon>`? Или элементом `<include>`? Почему не использовать вместо этого атрибут `include` или некоторую комбинацию значений `rel`? Почему ни одно, ни другое, ни третье, а все-таки элемент `<img>`? Очень просто: потому что Марк Андреессен (Marc Andreessen) реализовал его, а реализованный код побеждает.

Впрочем, нельзя сказать, что *весь* реализованный код побеждает; в конце концов, Andrew, Intermedia и HyTime тоже представляли собой реализованный код. Код — это необходимое, но недостаточное условие успеха. И я *действительно* не хочу сказать, что поставки кода еще до утверждения стандарта — это наилучшее решение. Введенный Марком элемент `<img>` не требовал общего графического формата; он не определял, каким образом этот элемент будет обтекаться текстом; он не поддерживал текстовых альтернатив или резервного, замещающего контента для более старых браузеров. И теперь, 17 лет спустя, мы продолжаем бороться с согласованием контента (content sniffing)[1], и данный элемент до сих пор продолжает оставаться источником безумных уязвимостей в системе безопасности (**http://code.google.com/p/doctype/wiki/ArticleContentSniffing**). И вы можете отследить всю эволюцию и развитие событий в исторической ретроспективе, вернувшись на 17 лет назад, в эпоху Великих браузерных войн (**http://en.wikipedia.org/wiki/Browser_wars**, **http://evolt.org/node/60181/**), до того момента, когда 25 февраля 1993 года Марк Андреессен (Marc Andreessen) небрежно заметил "возможно, когда-нибудь, MIME", а после этого все равно выпустил свой код.

Но для того, чтобы что-то победило, оно обязательно должно быть реализовано.

[1] См. **http://tools.ietf.org/html/draft-abarth-mime-sniff-06**.

Шкала развития HTML с 1997 по 2004 год

В декабре 1997 года консорциум WWW (World Wide Web Consortium, W3C) опубликовал стандарт HTML 4.0 (**http://www.w3.org/TR/REC-html40-971218/**), а вскоре после этого закрыл рабочую группу HTML (HTML Working Group). Менее чем через два месяца, отдельная рабочая группа W3C опубликовала спецификацию XML 1.0 **(http://www.w3.org/TR/1998/REC-xml-19980210**). А еще через три месяца руководство W3C провело семинар под названием "формирование будущего HTML" ("Shaping the Future of HTML", см. **http://www.w3.org/MarkUp/future/**) с тем, чтобы дать общественности ответ на вопрос "Не прекратил ли консорциум W3C работу над HTML?" Ответ на данный вопрос был таким:

В ходе дискуссий было решено, что дальнейшее расширение спецификации HTML 4.0 усложнится, как усложнится и преобразование приложений HTML 4.0 в приложения XML[1]. Предлагаемый выход из этой ситуации и освобождения от этих ограничений заключается в том, чтобы начать разработку следующего поколения HTML "с чистого листа", основываясь на наборах тегов XML.

Консорциум W3C принял решение переориентировать рабочую группу HTML на выработку "семейства наборов тегов XML". В качестве первого шага, предпринятого в декабре 1998, рассматривается черновик промежуточной спецификации, которая просто переформулирует HTML в термины XML без добавления новых элементов или атрибутов (**http://www.w3.org/TR/1998/WD-html-in-xml-19981205/**). Впоследствии эта спецификация получила известность под именем "XHTML 1.0" (**http://www.w3.org/TR/xhtml1/**). Она определила новый тип MIME для документов XHTML, `application/xhtml+xml`. Однако, чтобы упростить миграцию существующих страниц HTML 4, в состав этой спецификации было включено "Приложение C" (**http://www.w3.org/TR/xhtml1/#guidelines**), которое "обобщило руководящие указания по разработке для авторов, которые желали, чтобы их документы XHTML визуализировались существующими пользовательскими агентами HTML". В "Приложении C" говорилось о том, что авторам разрешается создавать так называемые "страницы XHTML", и при этом по-прежнему обслуживать их с помощью типа MIME `text/html`.

Следующей целью разработчиков спецификации явились Web-формы. В августе 1999 года та же рабочая группа HTML опубликовала первый черновик будущего стандарта XHTML Extended Forms (**http://www.w3.org/TR/1999/WD-xhtml-forms-**

---

[1] XML (eXtensible Markup Language) — расширяемый язык разметки, рекомендованный Консорциумом Всемирной паутины, и фактически представляющий собой свод общих синтаксических правил. Подробнее см. **http://en.wikipedia.org/wiki/XML**, **http://en.wikipedia.org/wiki/List_of_XML_markup_languages**, **http://www.codenet.ru/webmast/xml/**. XML — текстовый формат, предназначенный для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки, например, XHTML (**http://en.wikipedia.org/wiki/XHTML**). XML является упрощенным подмножеством языка SGML. — *Прим. перев.*

req-19990830). Свои ожидания они сформулировали в первом же абзаце (см. **http://www.w3.org/TR/1999/WD-xhtml-forms-req-19990830#intro**):

> "После всестороннего тщательного обсуждения, рабочая группа HTML приняла решение о том, что цели форм следующего поколения несовместимы с поддержанием обратной совместимости с браузерами, разрабатывавшимися для более ранних версий HTML. Наша цель заключается в предоставлении новой модели форм ("XHTML Extended Forms") на основе набора четко определенных требований. Требования, описанные в этом документе, основываются на опыте работы с широким спектром приложений, использующих формы".

Через несколько месяцев спецификация "XHTML Extended Forms" была переименована в "XForms", и для продолжения работы над ней была образована отдельная рабочая группа (**http://www.w3.org/MarkUp/Forms/2000/Charter.html**). Эта группа работала параллельно с рабочей группой HTML и, наконец, в октябре 2003 года опубликовала первую редакцию XForms 1.0 (**http://www.w3.org/TR/2003/REC-xforms-20031014/**).

Тем временем, когда завершился переход к XML, рабочая группа HTML сосредоточила свои усилия на создании "стандарта HTML нового поколения". В мае 2001 года они опубликовали первую редакцию стандарта XHTML 1.1 (**http://www.w3.org/TR/2001/REC-xhtml11-20010531/**), которая добавляла к XHTML 1.0 лишь небольшое количество дополнительных функций, имевших второстепенное значение, а также ликвидировала "Приложение C". Начиная с версии 1.1, все документы XHTML должны были обслуживаться с помощью типа MIME `application/xhtml+xml`.

Все, что вы знаете о XHTML — неправильно

Почему типы MIME настолько важны? Почему я все время к ним возвращаюсь? В трех словах: драконовская обработка ошибок (Draconian error handling). Браузеры всегда относились к HTML в высшей степени "терпимо". Если вы, создав страницу HTML, забудете использовать тег `</head>`, браузер все равно отобразит эту страницу. (Некоторые теги неявно подразумевают конец тега `<head>` и начало тега `<body>`.) Предполагается, что вы используете иерархическое вложение тегов — закрывая их по принципу "матрешки" (сначала наружный, затем внутренний). Но, если вы создадите разметку наподобие `<b><i></b></i>`, браузеры с ней уж как-нибудь да разберутся и продолжат работу без отображения сообщения об ошибке.

Как и следовало ожидать, тот факт, что "битая" разметка HTML все-таки работала в Web-браузерах, привел к тому, что авторы массово создавали "битые" страницы HTML. Таких страниц стало очень много! По некоторым оценкам, более 99% всех страниц HTML, существующих на сегодняшний день, содержат, как минимум, одну ошибку. Но, поскольку

эти ошибки не вызывают появления "ругательных" сообщений в браузерах, никто особенно не заботится об их исправлении.

Консорциум W3C счел это фундаментальной проблемой всей Web, и по этой причине его руководством было принято решение исправить данную ситуацию. Спецификация XML, опубликованная в 1997 году, разрушила традицию терпимости к ошибкам со стороны клиентов и ввела требование, гласящее, что все документы XML должны трактовать так называемые "ошибки формулировки" ("well-formedness" errors) как фатальные. Эта концепция "работы до первой ошибки" получила известность как "драконовская обработка ошибок", названная так в честь древнегреческого тирана по имени Дракон (Draco), правившего в Афинах и введшего законы, требовавшие смертной казни за любой, самый незначительный проступок (**http://en.wikipedia.org/wiki/Draco_%28lawgiver%29**). Когда консорциум W3C переформулировал HTML как "словарь" XML, было введено и требование, гласящее, что все документы, интерпретирующиеся с использованием нового типа MIME `application/xhtml+xml`, подлежат трактовке с драконовской обработкой ошибок. Даже если ваша страница XHTML содержала хоть единственную ошибку формулировки (well-formedness error), например, такую, как пропущенный тег `</head>`, или ошибку вложенности тегов начала и конца, то Web-браузеры больше не имели никакого выбора, кроме того, чтобы прекратить обработку страницы и вывести пользователю сообщение об ошибке.

Эта идея не завоевала популярности (**http://lists.w3.org/Archives/Public/w3c-sgml-wg/1997Apr/0164.html**, **http://blogs.msdn.com/b/xmlteam/archive/2005/06/17/430354.aspx**). С учетом того, что 99% существующих Web-страниц содержали хотя бы одну ошибку, возросшая вероятность отображения сообщений об ошибках конечным пользователям, тем самым ставящая крест на новых возможностях XHTML 1.0 и 1.1, привели к тому, что большинство авторов начало просто игнорировать новый тип `application/xhtml+xml`. Но это не означало, что авторы начали полностью игнорировать XHTML. В большинстве случаев они так не поступили. "Приложение C" спецификации XHTML 1.0 дало Web-разработчикам обходной путь: "использовать нечто, напоминающее синтаксис XHTML, но продолжать обрабатывать этот синтаксис с помощью типа MIME `text/html`". Именно по этому пути и пошли тысячи разработчиков Web: они "обновили" синтаксис до уровня XHTML но продолжали обрабатывать код с помощью типа MIME `text/html`.

Даже сегодня миллионы Web-страниц декларируются как страницы XHTML, их первая строка объявляет тип документа XHTML, в них используются имена тегов, написанные строчными буквами, атрибуты заключаются в кавычки, и в пустых элементах применяется завершающая наклонная косая черта, как, например, в `<br />` и `<hr />`. Но лишь небольшая часть этих страниц обрабатывается с помощью MIME-типа `application/xhtml+xml`, который и запускает "драконовскую" обработку ошибок, присущую XML. Любая страница, обслуживаемая с использованием MIME-типа `text/html` — вне зависимости от типа документа (doctype), синтаксиса или стиля кодирования, подвергается синтаксическому разбору "терпимого" анализатора HTML, и любые ошибки разметки тихо игнорируются, а предупреждения

пользователям (или кому-либо еще) никогда не выводятся, даже если технически страница некорректна.

Спецификация XHTML 1.0 включала обходной путь, но с выходом XHTML 1.1 он был закрыт, а так и не завершенная спецификация XHTML 2.0 продолжила традицию "драконовской обработки ошибок". Это и объясняет, почему существует такое гигантское количество страниц, которые объявлены как страницы XHTML 1.0, и так мало — страниц, объявленных как страницы XHTML 1.1 или XHTML 2.0. Поэтому задумайтесь, а действительно ли вы используете XHTML? Проверьте, какой MIME-тип применяется для разбора страниц. На самом деле, если вы точно не знаете используемого MIME-типа, то я почти уверен, что вы продолжаете работать с `text/html`. Если только вы не обслуживаете свои страницы с помощью MIME-типа `application/xhtml+xml`, то ваш так называемый "код XHTML" является XML-кодом лишь на словах.

Конкурирующий взгляд на будущее

В июне 2004 года консорциум W3C провел семинар-практикум по Web-приложениям и составным документам (**http://www.w3.org/2004/04/webapps-cdf-ws/**). На этом практическом семинаре присутствовали представители разработчиков трех наиболее популярных браузеров, ряда компаний, занимающихся Web-разработкой, а также другие члены консорциума. Группа заинтересованных сторон, включая Mozilla Foundation и Opera Software, представила собственный взгляд на будущее Web: эволюция существующего стандарта HTML 4 должна включить новые функции, необходимые разработчикам современных приложений Web (**http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html**).

По их представлениям, наиболее важное значение для последующего развития должны были иметь следующие семь принципов.

- Обратная совместимость, четкий путь миграции

  Технологии разработки Web-приложений должны основываться на технологиях, хорошо знакомых разработчикам приложений — HTML, CSS, DOM и JavaScript.

  Основные функции Web-приложений должны быть реализованы с помощью поведенческих механизмов (behaviors), скриптов и таблиц стилей, присутствующих на сегодняшний день в Internet Explorer 6, чтобы авторам предоставлялся четкий путь миграции. Любое решение, которое не может применяться в связке с современными популярными пользовательскими агентами (user agents) без необходимости прибегать к двоичным сборкам подключаемых модулей (plugins), имеет низкие шансы на успех.

- Четко определенная обработка ошибок

  Обработка ошибок в Web-приложениях должна быть определена на таком уровне детализации, когда браузерам не требуется вводить собственные механизмы обработки ошибок или, прибегая к методам обратной разработки, перенимать механизмы обработки ошибок, реализованные другими браузерами.

- Конечные пользователи не должны страдать от ошибок авторов

  Спецификации должны точно определять механизмы восстановления после сбоев для каждой конкретной ошибки. Большей частью обработка ошибок должна быть определена в терминологии корректного восстановления после ошибок (как это происходит в CSS), а не в виде катастрофических сбоев (как в XML).
- Практическое использование

  Каждая функция, которая входит в спецификацию Web-приложений, должна быть оправдана практическим применением. Справедливость обратного не гарантируется: каждый конкретный случай практического применения не обязательно должен гарантировать внесение новой функции в спецификации.

  Предпочтительно, чтобы примеры применения основывались на реальных сайтах, где авторы ранее использовали плохое решение, чтобы обойти ограничения.
- Решения на базе скриптов — это всерьез и надолго

  Но их следует избегать там, где вполне можно обойтись удобной декларативной разметкой.

  Решения на базе скриптов должны быть нейтральны по отношению к устройствам и презентации, за исключением случаев, когда они нацелены на конкретные устройства (например, включены в XBL[1]).
- Профилировки на конкретные устройства (Device-specific profiling) следует избегать

  Авторы должны иметь возможность полагаться на одни и те же функции, реализованные в вариантах пользовательского агента (User Agent, UA), ориентированных на настольные и на мобильные устройства.
- Открытость процесса

  Преимуществом Web всегда была открытая среда разработки. Web-приложения должны быть ядром Web, и их разработка тоже должна быть открытым процессом. Списки рассылки, архивы и черновики спецификаций постоянно должны быть доступны широкой публике.

В ходе опроса участникам семинара был задан вопрос: “Следует ли W3C развивать декларативные расширения для HTML и CSS, а также императивные расширения для DOM, с целью удовлетворения среднего уровня требований к Web-приложениям, в противовес сложным полнофункциональным API уровня операци-

---

[1] XBL (XML Binding Language) — это язык разметки на базе языка XML, использующийся для декларирования поведения и вида виджетов XUL (XML User Interface Language, язык разметки для создания динамических пользовательских интерфейсов, разрабатываемый в рамках проекта Mozilla, подробнее см. **http://en.wikipedia.org/wiki/XUL**, **http://ru.wikipedia.org/wiki/XULRunner**, **https://developer.mozilla.org/En/XUL**, **https://wiki.mozilla.org/XUL:Xul_Runner**) и элементов XML. XBL разрабатывался в рамках проекта Mozilla для использования в семействе приложений Mozilla (Mozilla Application Suite), на настоящий момент язык не описан ни одним формальным стандартом и является собственностью Mozilla. XBL 2.0 — это новая версия XBL, которая находится в процессе стандартизации Консорциумом мировой паутины. Подробнее см. **http://en.wikipedia.org/wiki/XBL**, **https://developer.mozilla.org/en/XBL**, **http://www.w3.org/TR/xbl/**. — *Прим. перев.*

онных систем?" Это предложение было внесено Яном Хиксоном (Ian Hickson), представителем Opera Software. В результате голосования 11 участников высказались "за" и 8 — "против". В своем заключительном документе по результатам семинара (**http://www.w3.org/2004/04/webapps-cdf-ws/summary**), консорциум отметил, что "в настоящее время W3C не намеревается выделять каких-либо ресурсов на третью тему опроса: расширения для HTML и CSS для Web-приложений, отличных от технологий, развиваемых в рамках контракта текущих рабочих групп W3C".

В условиях, когда было принято это решение, люди, предлагавшие развивать HTML и HTML-формы, имели только два варианта выбора дальнейших действий: отказаться от своих планов или же продолжить свою работу вне W3C. Они избрали второй путь и зарегистрировали домен whatwg.org, а в июне 2004 года появилась Рабочая группа WHAT (**http://www.whatwg.org/news/start**).

Рабочая группа WHAT

Что же представляет собой рабочая группа WHAT? Лучше всего дать объясниться самим членам этой организации:

"Рабочая группа WHAT (Web Hypertext Applications Technology Working Group) представляет собой открытую, неофициальную и свободную организацию разработчиков Web-браузеров и всех остальных заинтересованных сторон. Цель группы — разработка спецификаций на основе HTML и связанных с HTML технологий с целью упрощения развертывания интероперабельных Web-приложений и передачи результатов организациям, занимающимся стандартизацией. Такое сотрудничество должно способствовать формированию основы для формального расширения HTML и развития этой технологии в общем русле современных стандартов.

Создание данного форума проходило на фоне нескольких месяцев независимой работы и частного обмена почтовыми сообщениями с целью выработки спецификаций для таких новых технологий. Основное внимание было сфокусировано на задаче расширения форм HTML4 таким образом, чтобы обеспечить поддержку функций, запрошенных авторами, без нарушения обратной совместимости с существующим информационным содержимым. Данная группа была создана с тем, чтобы гарантировать, что дальнейшее развитие этих спецификаций будет полностью открытым и будет происходить через публично архивируемый список рассылки".

Ключевая фраза здесь — "без нарушения обратной совместимости". XHTML (за исключением "обходного пути", предоставляемого "Приложением C") не является стандар-

том, обратно совместимым с HTML. Новый стандарт требует абсолютно нового типа MIME и, кроме того, налагает обязательное требование "драконовской обработки ошибок" на все информационное содержимое, объявленное и поставляемое с указанным типом MIME. Стандарт XForms тоже не является обратно совместимым с формами HTML, потому что он может применяться только в документах, которые обслуживаются с новым MIME-типом XHTML, а это значит, что XForms тоже налагает требование "драконовской" обработки ошибок. Все дороги ведут к MIME.

Вместо того чтобы продираться через сложности, стоящие десятилетних инвестиций в HTML, а в результате сделать 99% существующих Web-страниц непригодными к использованию, рабочая группа WHAT решила пойти по иному пути: документировать "терпимые к ошибкам" алгоритмы, которые на практике применялись браузерами. Web-браузеры всегда лояльно относились к ошибкам HTML, но никто и никогда раньше не брал на себя труд описать, как именно они это делали. Браузер NCSA Mosaic имел собственные алгоритмы для обработки страниц с ошибками, и в Netscape тоже была сделана попытка организовать нечто похожее. Разработчики Internet Explorer, в свою очередь, подражали команде Netscape. А затем в Opera и Firefox опять была сделана попытка подражания — на этот раз, прототипом стал Internet Explorer. Наконец, разработчики Safari пытались реализовать похожие алгоритмы, взяв за образец Firefox. И эта цепочка подражаний тянется до сегодняшнего дня. По ходу дела разработчики тратили тысячи и тысячи часов рабочего времени, пытаясь добиться, чтобы их продукты были совместимы с продукцией их конкурентов.

Вам кажется, что это безумные временные затраты? Так оно и есть! Или, точнее говоря, так оно и было. На это потребовалось пять лет, но (если исключить всевозможные безумные экстремальные случаи) рабочая группа WHAT успешно документировала процесс разбора HTML (HTML parsing) таким образом, чтобы обеспечить совместимость с существующим информационным наполнением Web (**http://www.whatwg.org/specs/web-apps/current-work/multipage/parsing.html**). В окончательном варианте алгоритма нигде нет ни единого шага, который бы приводил к прекращению обработки и выводу сообщения об ошибке для конечного пользователя.

В течение всего времени, пока выполнялся этот титанический труд по обратной разработке (reverse engineering), рабочая группа WHAT, не привлекая к себе излишнего внимания и не афишируя свою деятельность, вела работу и над рядом других вопросов. Один из них представлял собой разработку спецификации, изначально названной Web Forms 2.0 (**http://www.whatwg.org/specs/web-forms/current-work/**). Эта спецификация добавила новые элементы управления к формам HTML. Впрочем, о Web-формах мы будем подробнее разговаривать далее, в *главе 9*, которая так и озаглавлена: *"Форма безумия"*. Еще одна задача заключалась в том, чтобы выработать спецификацию, получившую название "Web Applications 1.0" (**http://www.whatwg.org/specs/web-apps/current-work/complete/**), которая документировала основные новые функции наподобие элемента `<canvas>`, аналога "холста" для непосредственного рисования (этот элемент будет обсуждаться в *главе 4*), и встроенную поддержку для аудио и видео, без использования сторонних подключаемых модулей (plugins), речь о которой пойдет в *главе 5*.

Возврат к W3C

В течение двух с половиной лет консорциум W3C и рабочая группа WHAT практически полностью игнорировали друг друга. В то время как рабочая группа WHAT сконцентрировала все внимание на Web-формах и новых функциях HTML, рабочая группа HTML консорциума W3C занималась второй версией спецификации XHTML. Но к октябрю 2006 года стало ясно, что рабочая группа WHAT уже набрала серьезный потенциал, в то время пока спецификация XHTML 2 все еще прозябала на стадии черновика, а ее функции ни разу и никогда не были реализованы ни одним из ведущих разработчиков браузеров. В октябре 2006 года Тим Бернерс-Ли (Tim Berners-Lee), основатель консорциума W3C, объявил о том, что W3C начнет сотрудничать с рабочей группой WHAT в деле дальнейшего развития HTML (см. **http://dig.csail.mit.edu/breadcrumbs/node/166**). В частности, было объявлено о следующем:

"Некоторые вопросы по прошествии времени самопроизвольно прояснились. Так, стала очевидной необходимость пошагового развития HTML. Попытка принудительно "переключить" весь мир на использование XML, включая применение таких деталей, как использование кавычек для заключения в них значений атрибутов и символов наклонной косой черты при указании пустых тегов и пространств имен, очевидно, провалилась. Подавляющее большинство людей, создающих информационное наполнение в формате HTML, этого новшества не приняли, в основном, чтобы браузеры "не создавали осложнений". Некоторые крупные сообщества этот переход осуществили, и теперь наслаждаются результатом, полученным за счет использования четко сформулированных систем, но это и все. Очень важно развивать HTML пошагово, чтобы переход к четко сформулированному миру (well-formed world)[1] происходил не резко, а поэтапно. Кроме того, пошаговое развитие принесет и самому этому миру гораздо больше пользы.

[1] Термин *"Правильно построенный"* (Well-formed) является одним из ключевых в терминологии XML. Правильно построенный документ соответствует всем общим правилам синтаксиса XML, применимым к любому XML-документу. Если, например, начальный тег не имеет соответствующего ему конечного тега, то это *неправильно построенный* документ XML. Документ, который неправильно построен, не может считаться документом XML. XML-процессор (XML parser) не должен обрабатывать его обычным образом и обязан классифицировать ситуацию как фатальную ошибку. Такие ошибки еще называются "желтыми экранами смерти" (Yellow Screens of Death) — подробнее см. **http://commons.wikimedia.org/wiki/File:Yellow_screen_of_death.png**, **http://en.wikipedia.org/wiki/Screen_of_death#Yellow**, **http://en.wikipedia.org/wiki/XML#Well-formedness_and_error-handling**. — *Прим. перев.*

План заключается в формировании абсолютно новой рабочей группы HTML. В отличие от своей предшественницы, эта новая группа должна заниматься пошаговым наращиванием функциональности HTML, параллельно с развитием xHTML. Новая группа должна иметь нового руководителя и членов-разработчиков. Она должна уделять одинаковое внимание как HTML, так и xHTML, развивая их параллельно. Эта группа будет пользоваться мощной поддержкой со стороны всех членов сообщества, с которыми мы имели беседы, в том числе — с разработчиками браузеров.

Будет вестись и работа над формами. Это сложная область, так как и HTML-формы и XForms представляют собой языки описания форм. Формы HTML развертываются универсально, и существует множество реализаций XForms, нашедших себе разнообразные применения. Тем временем, переданный на обсуждение вариант спецификации Web Forms предлагает гибкие расширения для форм HTML. Общий план, по сообщению разработчиков, заключается в расширении функциональных возможностей форм HTML".

Одно из первых решений, принятых вновь сформированной рабочей группой W3C HTML, касается переименования спецификации "Web Applications 1.0" в "HTML5". С этого момента мы и начнем детальное изучение HTML5.

Эпилог

В октябре 2009 года консорциум W3C прекратил полномочия рабочей группы XHTML 2 (**http://www.w3.org/News/2009#item119**) и выступил с заявлением, обосновывающим это решение (**http://www.w3.org/2009/06/xhtml-faq.html**):

"Когда консорциум W3C делал заявление о рабочих группах HTML и XHTML 2 в марте 2007, мы указывали, что мы будем продолжать мониторинг рынка XHTML 2. W3C понимает важность программного заявления, сигнализирующего сообществу о будущем HTML.

Хотя мы высоко ценим вклад, внесенный рабочей группой XHTML 2, и работу, проделанную ею в течение многих лет, после длительного обсуждения со всеми участниками руководство W3C решило не продлевать полномочия этой рабочей группы после истечения срока их действия в 2009 году. Побеждает та технология, которая находит реализацию".

Рекомендованные материалы для дополнительного чтения

- "The History of the Web" (**http://hixie.ch/commentary/web/history**), старый материал об истории развития Web, написанный Яном Хиксоном (Ian Hickson).
- "HTML/History", документ об истории развития HTML (**http://www.w3.org/html/wg/wiki/History**), созданный Майклом Смитом (Michael Smith), Генри Сайвоненом (Henri Sivonen) и др.
- "A Brief History of HTML" (**http://atendesigngroup.com/blog/brief-history-html**), документ, кратко описывающий историю развития HTML и написанный Скоттом Райненом (Scott Reynen).

Глава 2

Определение поддержки функций HTML5

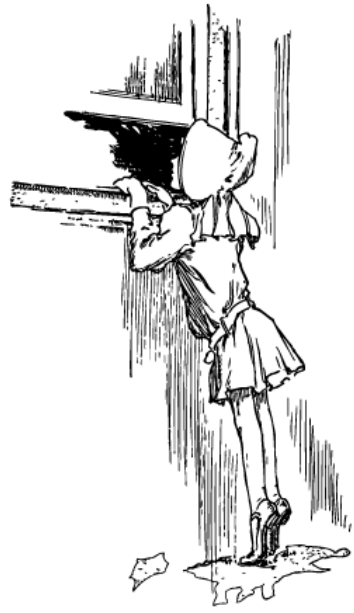
Возможно, у вас возникнет вопрос: "Как мне начать пользоваться HTML5, раз более старые браузеры его не поддерживают?" Но постановка этого вопроса некорректна, поскольку уводит нас в сторону от обсуждаемой темы. HTML5 не есть нечто единое и неделимое, напротив, это — набор самостоятельных функций. Поэтому, хотя вы и не можете определить, что есть "поддержка HTML5", поскольку данное определение не имеет смысла, вы все же можете выявить, поддерживает ли конкретный пользовательский агент (user agent)[1] отдельные функции HTML5, например, такие, как `<canvas>`, `<video>` или `<geolocation>`.

Методика выявления поддержки функций HTML5

Когда ваш браузер визуализирует Web-страницу, он строит коллекцию объектов Document Object Model (DOM)[2], которая представляет подборку элементов HTML, из которых состоит страница. Каждый элемент — любой тег, будь то `<p>`, `<div>`

[1] *Пользовательский агент* (User Agent) — это клиентское приложение, использующее определенный сетевой протокол. Как правило, этот термин обычно используется для приложений, осуществляющих доступ к Web-сайтам, а к таким приложениям относятся Web-браузеры, поисковые роботы (search engines) и другие "пауки", приложения, работающие на мобильных устройствах (смартфонах и др.). Подробнее см. **http://en.wikipedia.org/wiki/User_agent**, **http://ru.wikipedia.org/wiki/User_Agent**, а полный список существующих на настоящий момент "пользовательских агентов можно найти здесь: **http://www.user-agents.org/**. — *Прим. перев.*

[2] Объектная модель документа (Document Object Model, DOM) — это программный интерфейс, не зависящий от платформы и языка, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов. Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями родительский-дочерний. Изначально различные браузеры имели собственные модели документов (DOM), не совместимые с остальными. Для того чтобы обеспечить взаимную и обратную совместимость, специалисты консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM. Подробнее см. **http://ru.wikipedia.org/wiki/Document_Object_Model**, **http://en.wikipedia.org/wiki/Document_Object_Model**, **http://www.w3.org/DOM/**, **https://studio.tellme.com/dom/ref/**. — *Прим. перев.*



или — представляется в DOM соответствующим типом объекта. Имеются также и глобальные объекты, например, окна или документы, которые не привязаны к конкретным элементам разметки.

Все объекты DOM используют общий набор свойств, но некоторые объекты имеют больше свойств, чем другие. В браузерах, поддерживающих некоторые функции HTML5, некоторые объекты будут обладать уникальными свойствами. Беглый взгляд на модель DOM позволит вам выяснить, какие из функций HTML5 обеспечены поддержкой.

Существует четыре базовых метода, которые позволяют выяснить, поддерживает ли браузер ту или иную функцию HTML5. Если идти по пути "от простого к сложному", то эти методы формулируются так:

1. Сначала необходимо проверить, существует ли то или иное свойство в глобальном объекте (например, таком, как окно или навигатор). Пример: Тестирование поддержки элемента <geolocation> (пример на сайте автора книги: **http://diveintohtml5.org/detect.html#geolocation**).
2. Создайте элемент, а затем проверьте, обладает ли этот элемент нужным вам свойством. Пример: Тестирование поддержки элемента <canvas> (пример на сайте автора книги: **http://diveintohtml5.org/detect.html#canvas**).
3. Создайте элемент, проверьте, существует ли метод для данного элемента, затем вызовите данный метод и посмотрите, какое значение он возвращает. Пример: Проверка поддерживаемых форматов видео (пример на сайте автора книги: **http://diveintohtml5.org/detect.html#video-formats**).
4. Создайте элемент, установите для одного из его свойств конкретное значение, а затем проверьте, сохранилось ли это значение. Пример: Проверка поддерживаемых значений <input> (пример на сайте автора книги: **http://diveintohtml5.org/detect.html#input-types**).

Modernizr — библиотека выявления поддержки HTML5

Modernizr (**http://www.modernizr.com/**) представляет собой библиотеку JavaScript на основе открытого кода, лицензированную Массачусетским Технологическим институтом (MIT)[1], которая выявляет поддержку для многих функций HTML5 и CSS3 (**http://www.css3.info/**, **http://www.w3.org/TR/css3-roadmap/**).

[1] См. **http://web.mit.edu/, http://en.wikipedia.org/wiki/Massachusetts_Institute_of_Technology**, **http://tinyurl.com/6456dtl**.

На момент написания этих строк новейшей версией Modernizr была версия 1.5[1]. Общая рекомендация заключается в том, чтобы всегда пользоваться новейшей версией. Чтобы применять эту библиотеку, включите в начало вашей страницы элемент `<script>`, показанный в листинге 2.1 полужирным шрифтом. Выглядеть этот элемент должен следующим образом.

Листинг 2.1. Элемент `<script>`, который необходимо включить в начало Web-страницы для использования библиотеки Modernizr

```
<html>
<head>
  <meta charset="utf-8">
  <title>Dive Into HTML5</title>
  <script src="modernizr.min.js"></script>
</head>
<body>
  ...
</body>
</html>
```

`Modernizr` работает автоматически. Функции `modernizr_init()`, вызываемой для инициализации, нет. Когда библиотека запускается, она создает глобальный объект `Modernizr`, который содержит набор свойств типа `Boolean` для каждой функции, которую он может детектировать. Например, если ваш браузер поддерживает canvas API, который будет подробно рассматриваться в *главе 4*, то свойство `Modernizr.canvas` будет иметь значение `true`. Если ваш браузер не поддерживает canvas API, то свойство `Modernizr.canvas` будет иметь значение `false`. Пример детектирования поддержки canvas API продемонстрирован в листинге 2.2.

Листинг 2.2. Пример, демонстрирующий детектирование поддержки canvas API

```
if (Modernizr.canvas) {
  // можно попробовать начинать геометрические фигуры!
} else {
  // Встроенной поддержки элемента <canvas> браузер не обеспечивает :(
}
```

Элемент Canvas

HTML5 определяет элемент `<canvas>` как "элемент, представляющий собой область для отображения растровых изображений, зависимую от разрешения, которая может использоваться для оперативной ("на лету") визуализации графических изо-

[1] На момент подготовки русского издания — новейшей была версия 1.7, и это говорит о том, что библиотека постоянно развивается. Следите за обновлениями! — *Прим. перев.*

бражений, игровой графики и других изображений" (**http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html**). Этот элемент, так называемый "холст" (canvas), представляет собой прямоугольную область на вашей странице, в пределах которой вы, с помощью JavaScript, можете рисовать что угодно. HTML5 определяет набор функций (так называемый интерфейс прикладного программирования "холста", canvas API), предназначенных для рисования простых геометрических фигур, определения траекторий (paths), создания градиентов и применения трансформаций.

Проверка наличия поддержки canvas API использует метод выявления номер 2, описанный в *разд. "Методика выявления поддержки функций HTML5"*. Если ваш браузер поддерживает canvas API, то объект DOM, который он создает для представления элемента `<canvas>`, будет иметь метод `getContext()`. Если ваш браузер не поддерживает canvas API, то объект DOM, создаваемый им для элемента `<canvas>`, будет иметь только набор общих свойств, но ничего специфичного для элементов `<canvas>`. Проверку наличия поддержки canvas API можно выполнить с помощью функции `supports_canvas()`, показанной в листинге 2.3.

Листинг 2.3. Функция `supports_canvas()`

```
function supports_canvas() {
  return !!document.createElement('canvas').getContext;
}
```

Эта функция начинает с создания "болванки" элемента `<canvas>`. Но этот элемент никогда не присоединяется к вашей странице, поэтому его никто никогда не увидит. Созданный элемент будет "висеть" в памяти, никуда не попадая и ничего не отображая, примерно так же, как лодка тихо качается на реке с медленным течением.

```
return !!document.createElement('canvas').getContext;
```

Как только вы создадите "болванку" элемента `<canvas>`, вам следует протестировать наличие метода `getContext()`. Этот метод существует только в том случае, если ваш браузер поддерживает canvas API.

```
return !!document.createElement('canvas').getContext;
```

Наконец, вы должны использовать трюк с "двойным отрицанием", чтобы получить в результате значение типа `Boolean` (`true` или `false`).

```
return !!document.createElement('canvas').getContext;
```

Эта функция выявляет поддержку большинства функций canvas API, включая такие, как работу с геометрическими фигурами (shapes), траекториями (paths), градиентами (gradients) и орнаментами (patterns). Но эта функция не сможет выявить поддержку библиотеки `explorercanvas` (**https://code.google.com/p/explorercanvas/**) от сторонних разработчиков, которая реализует поддержку canvas API в Microsoft Internet Explorer.

Вместо того чтобы писать эту функцию самостоятельно, для выявления поддержки canvas API вы можете использовать `Modernizr`, как показано в листинге 2.4.

Листинг 2.4. Проверка поддержки элемента <canvas> с помощью Modernizr

```
if (Modernizr.canvas) {
  // можно попробовать начинать геометрические фигуры!
} else {
  // Встроенной поддержки элемента <canvas> браузер не обеспечивает :(
}
```

Существует и отдельный метод поддержки текста для canvas API, который я продемонстрирую далее.

Canvas Text

Даже если ваш браузер поддерживает canvas API, он не обязательно должен обеспечивать поддержку canvas text API, позволяющего добавлять текст на ваши "холсты". Интерфейс прикладного программирования canvas API развивался с течением времени, и текстовые функции были добавлены к нему относительно недавно. Некоторые браузеры уже включали поддержку элемента `<canvas>` на тот момент времени, когда текстовые функции еще не были добавлены в состав API.

Проверка наличия поддержки текстовых функций для canvas API использует методику проверки номер 2 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает canvas API, то DOM-объект, который он генерирует для представления элемента `<canvas>`, будет содержать метод `getContext()`. Если ваш браузер не поддерживает canvas API, то генерируемый им DOM-объект создаст элемент `<canvas>`, который обладает только базовым набором свойств, но не содержит функции, специфичной для элемента `<canvas>`. Для проверки наличия поддержки текстовых функций элемента `<canvas>`

Ваш браузер поддерживает Canvas text API

можно использовать функцию `supports_canvast_text()`, показанную в листинге 2.5.

Листинг 2.5. Функция `supports_canvast_text()` для проверки наличия поддержки текстовых функций элемента `<canvas>`

```
function supports_canvas_text() {
  if (!supports_canvas()) { return false; }
  var dummy_canvas = document.createElement('canvas');
  var context = dummy_canvas.getContext('2d');
  return typeof context.fillText == 'function';
}
```

Эта функция начинается с проверки наличия поддержки элемента `<canvas>` с помощью функции `supports_canvas()`, которая уже рассматривалась в предыдущем разделе. Если ваш браузер не поддерживает canvas API, то он уж точно не поддерживает и canvas text API!

```
if (!supports_canvas()) { return false; }
```

Далее, вы создаете "болванку" элемента `<canvas>` и получаете ее контекст рисования. Это гарантированно должно работать, потому что функция `supports_canvas()` уже выполнила проверку существования метода `getContext()` для всех объектов элемента `<canvas>`.

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

Наконец, вы проверяете, имеет ли в контексте рисования функция `fillText()`. Если это так, значит, текстовые функции для canvas API доступны. Ура!

```
  return typeof context.fillText == 'function';
```

Вместо самостоятельного написания этой функции вы можете использовать библиотеку `Modernizr` для выявления поддержки текстовых функций canvas API, как показано в листинге 2.6.

Листинг 2.6. Проверка поддержки текстовых функций canvas API

```
if (Modernizr.canvastext) {
  // можно попытаться начать вводить текст!
} else {
  // браузер не обеспечивает встроенной поддержки для текстовых функций :(
}
```

Видео

HTML5 определяет новый элемент, который называется `<video>` и предназначен для встраивания видеороликов в ваши Web-страницы. Раньше встраивание видео было невозможным без применения плагинов от сторонних разработчиков, например, таких как Apple QuickTime® или Adobe Flash®.

Элемент `<video>` разработан таким образом, чтобы его можно было применять без каких-либо скриптов, выполняющих проверку поддержки видео. Вы можете указать множество видеофайлов, и браузеры, обеспечивающие поддержку видеофункций HTML5, выберут одну из этих функций, основываясь на том, какие видеоформаты они поддерживают. Чтобы узнать больше о различных форматах видео, см. статью о видео, озаглавленную "Простое введение в кодирование видео" ("A gentle introduction to video encoding"). Статья состоит из двух частей: части 1, посвященной контейнерным форматам (**http://diveintomark.org/archives/2008/12/18/give-part-1-container-formats**) и части 2, посвященной видеокодекам, выполняющим кодирование с потерями (**http://diveintomark.org/archives/2008/12/19/give-part-2-lossy-video-codecs**).

Ваш браузер поддерживает видео по стандарту HTML5

Браузеры, которые не поддерживают видеофункции HTML5, будут полностью игнорировать элемент `<video>`, но вы, тем не менее, можете обернуть эту особенность себе во благо, дав таким браузерам указание воспроизводить видео с помощью плагинов сторонней разработки. Крок Кеймен (Kroc Camen) разработал решение, названное им "Видео для всех" (Video for Everybody!, см. **http://camendesign.com/code/video_for_everybody**), которое использует элемент `<video>` стандарта HTML5 везде, где это возможно, но прибегает к плагинам QuickTime или Flash, если браузер не поддерживает функций HTML5 по работе с видео. Это решение совсем не использует JavaScript и работает практически с любым браузером, включая даже браузеры, предназначенные для мобильных устройств.

Если вы хотите получить дополнительные возможности по работе с видео, а не только помещать видеоролики на свои страницы и воспроизводить их, то вам уже потребуется JavaScript. Проверка наличия поддержки видео использует метод детекции номер 2 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает функцию видео HTML5, то DOM-объект, который он создает для представления элемента `<video>`, будет иметь метод `canPlayType()`. Если ваш браузер не поддерживает видео в соответствии со стандартом HTML5, то DOM-объект, создаваемый для элемента `<video>`, будет иметь только набор свойств, общий для всех элементов. Проверить поддержку видео можно с помощью функции `supports_video()`, представленной в листинге 2.7.

Листинг 2.7. Функция `supports_video()`

```
function supports_video() {
  return !!document.createElement('video').canPlayType;
}
```

Вместо самостоятельного написания этой функции вы можете использовать `Modernizr` для выявления поддержки видео в соответствии со стандартом HTML5, как показано в листинге 2.8.

Листинг 2.8. Выявление поддержки видео в соответствии со стандартом HTML5 с помощью библиотеки `Modernizr`

```
// проверка поддержки видео HTML5
if (Modernizr.video) {
  // можно попытаться воспроизвести видеоролик!
} else {
  // браузер не обеспечивает встроенной поддержки видео :(
  // попробуйте воспользоваться QuickTime или Flash
}
```

В *главе 5*, посвященной видео, я расскажу и о другом решении, которое тоже использует обсуждаемые методики выявления поддержки функций HTML5 для преобразования элементов `<video>` в форматы, пригодные для воспроизведения видеопроигрывателями на основе Flash, что даст возможность воспроизведения видеороликов с помощью браузеров, не поддерживающих видеофункций HTML5.

Существует и отдельный тест на определение поддерживаемых видеоформатов, которые могут воспроизводиться в вашем браузере. Этот тест я опишу в следующем разделе.

Форматы видео

Работа с видеоформатами в чем-то напоминает письменный перевод с одного языка на другой. Публикация в англоязычной версии газеты может содержать ту же самую информацию, что и публикация в испаноязычной версии той же самой (или другой) газеты, но если вы можете читать только на одном из этих языков, то только одна из этих публикаций будет вам полезна! Чтобы воспроизвести видеоролик, ваш браузер должен "понимать" тот "язык", с помощью которого этот ролик был закодирован и записан.

"Языки" кодирования и записи видеофайлов называются "кодеками" (codecs, coder/decoder). Кодеки представляют собой алгоритмы для перекодирования видео в потоки битов. Всего в природе существует несколько десятков кодеков. Какой из них следует применять вам? К сожалению, неприятная реальность заключается в том, что видео по стандарту HTML5 — это не та область, в которой все браузеры могут соблюдать общее соглашение о едином, универсально поддерживаемом кодеке. Однако этот список удалось сузить до двух кодеков. Один из них является платным (потому что реализован на основе запатентованной технологии), но зато он работает с браузером Safari (**http://www.apple.com/safari/**) и на таких устройствах, как iPhone. Он же будет работать

Ваш браузер может воспроизводить видео в формате Ogg Theora, но не поддерживает видео в формате H.264

и с Flash, если вы будете использовать решения наподобие Video for Everybody! (**http://camendesign.com/code/video_for_everybody**). Другой же кодек бесплатен и работает с браузерами, разработанными на основе открытого исходного кода, такими, как Chromium (**https://code.google.com/chromium/**) и Mozilla Firefox (**https://www.mozilla.com/ru/firefox/?from=getfirefox**).

Проверка поддержки видеоформата осуществляется с использованием методики детекции номер три (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает видео в соответствии со стандартом HTML5, то DOM-объект, создаваемый им для представления элемента `<video>`, будет иметь метод `canPlayType()`. Этот метод сообщит вам, поддерживает ли браузер конкретный формат видео.

Следующая функция, представленная в листинге 2.9, осуществляет проверку поддержки патентованных форматов видео для компьютеров Mac и устройств типа iPhone.

Листинг 2.9. Функция для проверки поддержки видеоформатов для продукции Apple

```
function supports_h264_baseline_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
}
```

Эта функция начинает выполнение с проверки поддержки видео по стандарту HTML5, используя функцию `supports_video()`, которая уже обсуждалась в предыдущем разделе. Если ваш браузер не поддерживает видео по стандарту HTML5, он точно не поддерживает и ни одного из видеоформатов!

```
  if (!supports_video()) { return false; }
```

Затем функция создает "болванку" для элемента `<video>` (но не связывает ее со страницей, так что "болванка" останется невидимой) и вызывает метод `canPlayType()`. Этот метод присутствует гарантированно, потому что только что была выполнена проверка функции `supports_video()` на его наличие.

```
  var v = document.createElement("video");
```

"Видеоформат" — это, на самом деле, комбинация нескольких различных вещей. Говоря техническими терминами, вы спрашиваете у браузера, может ли он воспроизводить видео H.264 Baseline и аудио AAC LC в контейнере MPEG-4.

```
  return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
```

Примечание

О том, что означают все эти термины, будет рассказано в *главе 5*, посвященной видео. Возможно, вас также заинтересует ранее уже упомянутая статья *"Простое введение в кодирование видео"* ("A gentle introduction to video encoding").

Функция `canPlayType()` не возвращает значений `true` или `false`. При определении сложности видеоформатов функция возвращает строку:

- `"probably"`, если браузер с уверенностью может воспроизвести указанный формат;
- `"maybe"`, если существует вероятность того, что браузер справится с воспроизведением;
- `""` (пустая строка), если браузер явно не может воспроизвести этот формат.

Вторая функция, представленная в листинге 2.10, проверяет открытый видеоформат, поддерживаемый Mozilla Firefox и другими браузерами на основе открытого кода. Этот процесс почти полностью аналогичен первому; единственное отличие заключается в строке, которую вы передаете функции `canPlayType()`. Говоря технически, вы спрашиваете браузер, может ли он воспроизводить видео Theora и аудио Vorbis в контейнере Ogg.

Листинг 2.10. Функция, проверяющая поддержку видеоформата Mozilla Firefox и другими браузерами на основе открытого кода

```
function supports_ogg_theora_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/ogg; codecs="theora, vorbis"');
}
```

Наконец, WebM (**http://www.webmproject.org/**) представляет собой новый видеокодек, недавно переведенный в разряд открытых (и необремененных патентами), который будет включен в следующую версию большинства современных браузеров, включая Chrome, Firefox (**http://nightly.mozilla.org/**) и Opera (**http://labs.opera.com/news/2010/05/19/**). Для выявления поддержки видео WebM можно использовать тот же самый метод, как показано в листинге 2.11.

Листинг 2.11. Функция, проверяющая поддержку видеокодека WebM

```
function supports_webm_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/webm; codecs="vp8, vorbis"');
}
```

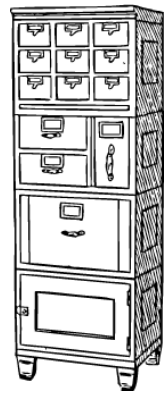
Вместо самостоятельного написания этой функции, можно применять `Modernizr` (версии 1.5 или более новой), чтобы выявлять поддержку браузером форматов видео по стандарту HTML5. Проверка поддержки видеоформатов HTML5 с помощью `Modernizr` представлена в листинге 2.12.

Листинг 2.12. Проверка поддержки видеоформатов HTML5 с помощью Modernizr

```
if (Modernizr.video) {
  // Давайте, воспроизведем видео! Но в каком формате?
  if (Modernizr.video.webm) {
    // Попробуем WebM
  } else if (Modernizr.video.ogg) {
    // Попробуем Ogg Theora + Vorbis в контейнере Ogg
  } else if (Modernizr.video.h264){
    // Попробуем H.264 video + AAC audio в контейнере MP4
  }
}
```

Локальное хранилище

Локальное хранилище HTML5 (**http://dev.w3.org/html5/webstorage/**) предоставляет способ, пользуясь которым Web-сайты могут хранить информацию на вашем компьютере с возможностью последующего ее извлечения. Используемая концепция аналогична концепции cookie-файлов, но позволяет хранить большие объемы информации. Cookie-файлы ограничены в размерах, и ваш браузер отсылает их назад на Web-сервер каждый раз, когда запрашивает новую страницу (что требует дополнительного времени и поглощает драгоценные ресурсы полосы пропускания). Хранилище HTML5 остается на вашем компьютере, и Web-сайты могут получать к нему доступ через JavaScript после загрузки страницы.



Вопрос к Профессору Разметки

В: Действительно ли локальное хранилище (local storage) — это часть HTML5? Почему для него отвели отдельную спецификацию?

О: Краткий ответ звучит так: да, локальное хранилище действительно является частью HTML5. Чуть более подробный ответ будет выглядеть так: локальное хранилище являлось частью основной спецификации HTML5, но было выделено в отдельную спецификацию, потому что некоторым членам рабочей группы HTML5 показалось, что объем общей спецификации HTML5 начал разрастаться за пределы разумного. В целом, поступок рабочей группы в отношении локального хранилища выглядит как обычный житейский поступок — пирог слишком велик, поэтому давайте разрежем его на несколько частей… В общем, добро пожаловать в "скользкий" мир стандартов.

Проверка наличия поддержки браузером локального хранилища HTML5 использует методику детекции номер 1 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает локальное хранилище HTML5, то глобальный оконный объект должен иметь свойство `localStorage`. Если ваш браузер такой поддержки не обеспечивает, то свойство `localStorage` не будет определено. К сожалению, крайне неприятный баг в старых версиях Firefox приводит

к возникновению исключения, если блокированы cookie-файлы, поэтому весь тест заключается в утверждение `try..catch`, как показано в листинге 2.13.

Листинг 2.13. Функция supports_local_storage(), предназначенная для проверки поддержки браузером локального хранилища по стандарту HTML5

```
function supports_local_storage() {
  try {
    return 'localStorage' in window && window['localStorage'] !== null;
  } catch(e){
    return false;
  }
}
```

Вместо самостоятельного написания этой функции вы можете воспользоваться библиотекой `Modernizr` (версии 1.1 или более новой) для выявления поддержки локального хранилища HTML5, как показано в листинге 2.14.

Листинг 2.14. Проверка поддержки локального хранилища HTML5 с помощью библиотеки `Modernizr`

```
if (Modernizr.localstorage) {
  // window.localStorage is available!
} else {
  // no native support for local storage :(
  // maybe try Gears or another third-party solution
}
```

Обратите внимание на чувствительность JavaScript к регистру символов. Атрибут `Modernizr` называется `localstorage` (все символы строчные), но свойство DOM называется `window.localStorage` (смешанный регистр).

Вопрос к Профессору Разметки

В: Насколько хорошо защищена моя база данных хранилища HTML5? Может ли кто-нибудь ее прочесть?

О: Просмотреть вашу базу данных HTML5 (и даже внести в нее изменения) может любой, кто получит физический доступ к вашему компьютеру. В тех пределах, которые представляет ваш браузер, любой Web-сайт может читать и модифицировать собственные значения, но сайты не могут читать и модифицировать значения, записанные другими сайтами. Это называется правилом ограничения домена (same-origin restriction)[1] (**http://www.whatwg.org/specs/web-apps/current-work/multipage/origin-0.html#origin-0**).

[1] Правило ограничения домена (same-origin policy, same-origin restriction) — это важная концепция безопасности для некоторых языков программирования на стороне клиента, например, JavaScript. В пределах одного сайта политика разрешает сценариям, находящимся на страницах одного сайта, доступ к методам и свойствам друг друга без ограничений, но предотвращает доступ к большинству методов и свойств для страниц на разных сайтах. Подробнее см. **http://en.wikipedia.org/wiki/Same_origin_policy**, **http://en.wikipedia.org/wiki/Cross-site_scripting**. — *Прим. перев.*

Web Workers

Web Workers — это стандартный метод, который используется браузерами для запуска JavaScript в фоновом режиме. С помощью Web Workers (**http://www.whatwg.org/specs/web-workers/current-work/**) вы сможете размножать "нити управления" (threads), работающие параллельно. Задумайтесь о том, как организована многозадачность на вашем компьютере, и вы окажетесь на правильном пути. Эти "фоновые нити управления" или потоки могут выполнять сложные математические расчеты, выполнять сетевые запросы или получать доступ к локальному хранилищу в то время, как основная Web-страница реагирует на действия пользователя — прокрутку содержимого, щелчки мышью или ввод информации с клавиатуры.

Проверка поддержки Web Workers использует метод детекции номер 1 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает Web Worker API, то глобальный объект `window` будет иметь свойство `Worker`. Если ваш браузер не поддерживает Web Worker API, то свойство `Worker` будет не определено.

Проверка поддержки Web Worker API вашим браузером продемонстрирована в листинге 2.15.

Листинг 2.15. Фукнция, проверяющая поддержку Web Workers

```
function supports_web_workers() {
  return !!window.Worker;
}
```

Вместо самостоятельного написания этой функции, вы можете использовать `Modernizr` (версии 1.1 или более новой), чтобы выявить поддержку Web Workers, как показано в листинге 2.16.

Листинг 2.16. Проверка поддержки Web Workers с помощью `Modernizr`

```
if (Modernizr.webworkers) {
  // Доступно свойство window.Worker!
} else {
  // Встроенной поддержки для Web Workers нет :(
  // Стоит попробовать Gears или какое-нибудь еще стороннее решение
}
```

Обратите внимание, что язык JavaScript чувствителен к регистру символов. Атрибут `Modernizr` называется `webworkers` (все символы строчные), но DOM-объект называется `window.Worker` (в слове "Worker" используется заглавная буква "W").

Автономные Web-приложения

Чтение статичных Web-страниц в автономном режиме (offline) не представляет большой проблемы: подключитесь к Интернету, загрузите Web-страницу, отключитесь от Интернета, затем вы сможете переместиться в изолированное помещение и там читать Web-страницу в свое удовольствие. (Чтобы сэкономить время, можно и никуда не перемещаться.) Но как быть с такими Web-приложениями, как Gmail или Google Docs? Благодаря HTML5, любой разработчик (не только Google!) может создавать Web-приложения, способные работать автономно.

Автономные Web-приложения (**http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html#offline**) запускаются в онлайновом режиме. Когда вы впервые посещаете Web-сайт, предоставляющий возможности работы в автономном режиме, Web-сервер сообщает браузеру о том, какие файлы ему нужны для работы в автономном режиме. Эти файлы могут представлять собой что угодно — HTML-документы, файлы JavaScript, изображения и даже видео. Как только ваш браузер загрузит все необходимые файлы, вы сможете работать с Web-сайтом, даже не имея соединения с Интернетом. Ваш браузер обнаружит, что вы не имеете интернет-подключения, и перейдет на использование уже загруженных файлов. Когда вы снова установите интернет-соединение, все изменения, внесенные вами во время автономной работы, могут быть закачаны на удаленный Web-сервер.

Ваш браузер поддерживает автономные Web-приложения

Проверка наличия поддержки режима автономной работы использует метод детекции номер 1 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает автономную работу с Web-приложениями, то глобальный объект `window` будет иметь свойство `applicationCache`. Если же браузер не поддерживает автономную работу с Web-приложениями, то свойство `applicationCache` не будет определено. Проверить наличие поддержки автономного режима работы с Web-приложениями можно с помощью следующей функции, представленной в листинге 2.17.

Листинг 2.17. Функция, выявляющая поддержку автономного режима работы с Web-приложениями

```
function supports_offline() {
  return !!window.applicationCache;
}
```

Вместо самостоятельного написания этой функции можно воспользоваться библиотекой `Modernizr` (версии 1.1 или более новой), как показано в листинге 2.18.

Листинг 2.18. Проверка наличия поддержки режима автономной работы с помощью `Modernizr`

```
if (Modernizr.applicationcache) {
  // Доступно свойство window.applicationCache!
} else {
  // Нет встроенной поддержки автономной работы с Web-приложениями :(
  // Возможно, стоит попробовать Gears или еще какое-нибудь стороннее решение
}
```

Обратите внимание на то, что язык JavaScript чувствителен к регистру символов, вводимых с клавиатуры. Атрибут `Modernizr` называется `applicationcache` (все буквы в этом имени — строчные), но DOM-объект называется `window.applicationCache` (смешанный регистр).

Географическое местоположение (Geolocation)

Элемент `<geolocation>` предоставляет метод определения географических координат вашего местоположения и (при желании) предоставления этой информации тем людям, которым вы доверяете. Это — не только метод определения вашего географического местоположения, потому что определяются ваш IP-адрес, ваше беспроводное сетевое соединение, информация о вышке сотовой связи, к которой обращается ваш мобильный телефон, а также сведения об оборудовании GPS, которое определяет географические координаты (широту и долготу) вашего местоположения, отсылаемую спутниками связи.

Вопрос к Профессору Разметки

В: Является ли элемент `<geolocation>` частью HTML5? Почему вы о нем упоминаете?

О: Поддержка элемента `<geolocation>` как раз сейчас добавляется в браузеры, наряду с поддержкой новых функций HTML5. Строго говоря, элемент `<geolocation>` стандартизуется рабочей группой Geolocation (**http://www.w3.org/2008/geolocation/**), которая работает независимо от рабочей группы HTML5. Но я в этой книге опишу элемент `<geolocation>`, потому что его введение тоже является важным шагом в ходе преобразований Web, происходящих в настоящий момент.

Проверка наличия поддержки элемента `geolocation` использует методику детектирования номер 1 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает geolocation API, то глобальный объект навигации (`navigator`) будет иметь свойство `geolocation`. Если же браузер такой поддержки не обеспечивает, то свойство `geolocation` будет не определено. Проверка поддержки элемента `geolocation` осуществляется так, как показано в листинге 2.19.

Листинг 2.19. Выявление поддержки элемента `geolocation`

```
function supports_geolocation() {
  return !!navigator.geolocation;
}
```

Вместо самостоятельного написания этой функции, можно для определения поддержки geolocation API воспользоваться библиотекой `Modernizr`, как показано в листинге 2.20.

Листинг 2.20. Проверка поддержки geolocation API с помощью `Modernizr`

```
if (Modernizr.geolocation) {
  // Попробуем определить, где мы находимся!
} else {
  // Встроенной поддержки геолокации нет :(
  // Возможно, стоит попробовать Gears или еще какое-нибудь стороннее решение
}
```

Если ваш браузер не обеспечивает встроенной поддержки geolocation API, надежда все еще остается. Gears (**http://tools.google.com/gears/**) — это плагин для браузеров на основе открытого кода, разработанный Google. Этот плагин работает под Windows, Mac, Linux, Windows Mobile и Android. Он предоставляет более старым браузерам все новые функции, обсуждавшиеся ранее в данной главе. Одной из функций, предоставляемых плагином Gears, как раз и является поддержка geolocation API. Это — не то же самое, что и поддержка `navigator.geolocation` API, но обе функции служат одной и той же цели.

На старых платформах мобильных телефонов тоже есть функции поддержки geolocation API, специфичные для конкретных платформ, в том числе BlackBerry (**http://www.tonybunce.com/2008/05/08/Blackberry-Browser-Amp-GPS.aspx**), Nokia (**http://wiki.forum.nokia.com/index.php/Bondi_Widget_porting_example_-_geolocation_API**), Palm (**http://developer.palm.com/index.php?option=com_content&view=article&id=1673#GPS-getCurrentPosition**) и OMTP BONDI (**http://bondi.omtp.org/1.0/apis/geolocation.html**).

В *главе 6*, полностью посвященной вопросам геолокации, мы в исчерпывающих подробностях рассмотрим использование всех этих разнообразных API.

Типы ввода

Вы знаете все о Web-формах, не так ли? Нужно создать элемент `<form>`, добавить несколько элементов `<input type="text">` и, возможно, элемент `<input type="password">`, а затем завершить форму кнопкой-элементом `<input type="submit">`.

Если это и все, что вы знаете, значит, вы не знаете даже половины того, что доступно вам теперь при создании форм. HTML5 определяет еще дюжину новых типов ввода, которые вы можете использовать в своих формах.

1. `<input type="search">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#text-state-and-search-state**) для полей поиска
2. `<input type="number">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/number-state.html#number-state**) для полей численного ввода с прокруткой (spinboxes)
3. `<input type="range">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/number-state.html#range-state**) для ползунковых регуляторов (sliders)
4. `<input type="color">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/number-state.html#color-state**) для инструментов выбора цвета (color pickers)
5. `<input type="tel">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#telephone-state**) для ввода телефонных номеров
6. `<input type="url">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#url-state**) для Web-адресов
7. `<input type="email">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#e-mail-state**) для адресов электронной почты
8. `<input type="date">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#date-state**) для инструментов выбора календарной даты
9. `<input type="month">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#month-state**) для выбора месяца
10. `<input type="week">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#week-state**) для выбора недели
11. `<input type="time">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#time-state**) для временных отметок (timestamps)
12. `<input type="datetime">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#date-and-time-state**) для ввода точной, абсолютной даты и точного времени
13. `<input type="datetime-local">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#local-date-and-time-state**) для ввода локальной даты и времени

Проверка поддержки типов вводимой информации по стандарту HTML5 использует методику детекции номер 4 (см. *разд. "Методика выявления поддержки*

функций HTML5"). Сначала вам нужно создать в памяти "болванку" элемента `<input>`. Для всех типов ввода `<input>` по умолчанию используется элемент `"text"`. Это критически важно.

```
var i = document.createElement("input");
```

Затем вам необходимо установить для элемента `<input>` атрибут типа, поддержку которого вам требуется выявить.

```
i.setAttribute("type", "color");
```

Если ваш браузер поддерживает этот конкретный тип ввода, то свойство `type` получит указанное вами значение. Если ваш браузер не поддерживает этот конкретный тип ввода, он проигнорирует установленное вами значение, и свойство `type` сохранит значение `"text"`.

```
return i.type !== "text";
```

Вместо самостоятельного написания 13 отдельных функций, для выявления поддержки новых типов ввода, определенных в HTML5, можно воспользоваться библиотекой `Modernizr`. `Modernizr` использует единственный элемент `<input>` для эффективного определения поддержки всех 13 новых типов ввода. Затем строится новый хэш с именем `Modernizr.inputtypes`, который содержит 13 ключей (атрибутов типов HTML5) и 13 значений типа `Boolean` (значение `true` указывает на наличие поддержки, а `false` — соответственно, на ее отсутствие).

Проверка наличия встроенной поддержки для инструмента ввода даты (date picker) представлена в листинге 2.21.

Листинг 2.21. Проверка наличия встроенной поддержки для инструмента выбора календарной даты

```
if (!Modernizr.inputtypes.date) {
  // нет встроенной поддержки для <input type="date"> :(
  // возможно, стоит попытаться построить собственный
  // с помощью Dojo или jQueryUI
}
```

Возможно, если поддержки нужного вам типа ввода нет, стоит попытаться создать собственный с помощью Dojo (**http://docs.dojocampus.org/dojox/widget/Calendar**) или jQueryUI (**http://jqueryui.com/demos/datepicker/**).

Текстовые заполнители (Placeholder Text)

Помимо новых типов ввода, HTML5 предлагает некоторые возможности по настройке существующих форм. Одним из улучшений является возможность установки текста-заполнителя в поле ввода. Текст-заполнитель отображается в поле ввода до тех пор, пока это поле еще не получило фокуса ввода и пользователь еще не начинал вводить в него свою информацию. Как только пользователь выполняет на этом поле щелчок мышью или переходит в него нажатием клавиши <Tab>,

текст-заполнитель исчезает. В главе, посвященной Web-формам, будут показаны экранные снимки ситуаций, когда у вас возникают проблемы с визуализацией текста-заполнителя.

Проверка поддержки текста-заполнителя использует методику детекции номер 2 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает функцию текста-заполнителя в полях ввода, то DOM-объект, создаваемый для представления элемента `<input>`, будет иметь свойство `placeholder` (даже если вы не включите в свой документ HTML атрибута `placeholder`). Если ваш браузер не поддерживает функцию текста-заполнителя, то DOM-объект, который он создаст для элемента `<input>`, не будет иметь свойства `placeholder`. Функция, помогающая выявить поддержку текста-заполнителя, представлена в листинге 2.22.

Листинг 2.22. Функция `supports_input_placeholder()`

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

Вместо самостоятельного написания этой функции, вы можете использовать библиотеку `Modernizr` (версии 1.1 или более новой) для выявления поддержки отображения текста-заполнителя, как показано в листинге 2.23.

Листинг 2.23. Проверка поддержки отображения текста-заполнителя с помощью `Moderniz`

```
if (Modernizr.input.placeholder) {
  // Ваш текст-заполнитель должен быть видимым!
} else {
  // поддержки текста-заполнителя нет :(
  // попробуйте решить проблему с помощью скрипта
}
```

Автофокус формы

Ваш браузер не поддерживает автофокуса формы

Web-сайты могут использовать JavaScript для автоматической фокусировки на первом поле формы, предназначенном для ввода информации. Например, домашняя страница сайта Google.com выполняет автоматическую фокусировку на поле ввода таким образом, чтобы вы могли сразу же начинать ввод ключевых слов для поиска, без необходимости самостоятельно помещать курсор в поле поиска. Хотя для большинства пользователей это удобно, некоторых этот "навязчивый сервис" может раз-

дражать — особенно это относится к людям, имеющим особые потребности, и опытным пользователям. Если вы нажмете клавишу пробела, ожидая прокрутки страницы, то прокрутки не произойдет, потому что курсор уже находится в поле ввода формы. Фактически, вместо прокрутки вы введете в поле ввода символ пробела. Если вы передадите фокус ввода другому полю в то время, пока страница все еще продолжает загружаться, скрипт автоматической фокусировки сайта "услужливо" переместит фокус ввода обратно в исходное поле ввода, в результате чего начатый вами ввод попадет не в то поле, в которое вы собирались вводить информацию. Поскольку автоматическая фокусировка выполняется с помощью JavaScript, обход всех этих "острых углов" может начать представлять проблему, особенно для тех людей, кто не хочет, чтобы приложение или Web-страница "похищала" у них фокус ввода.

Чтобы решить эту проблему, HTML5 вводит атрибут `autofocus` для всех элементов управления Web-формы (**http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#autofocusing-a-form-control**). Атрибут `autofocus` не означает в точности того, на что намекает его название: на самом деле этот атрибут передает фокус ввода конкретному полю, предназначенному для ввода информации. Но, поскольку это всего лишь разметка, а не скрипт, его поведение будет последовательным и логичным на всех Web-сайтах. Кроме того, поставщики браузеров (как и разработчики расширений для браузеров) могут предложить пользователям способы блокировки функции автоматического фокусирования.

Проверка наличия поддержки функции автоматической фокусировки по стандарту HTML5 использует методику детектирования номер 2 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает автоматическую фокусировку на элементах управления Web-формы, то DOM-объект, который он создаст для представления элемента `<input>`, будет иметь свойство `autofocus` (даже если вы не включали атрибут `autofocus` в ваш HTML-код). Если же ваш браузер не поддерживает автоматической фокусировки на элементах управления Web-формы, то DOM-объект, создаваемый для элемента `<input>`, не будет иметь свойства `autofocus`. Определить наличие поддержки для функции автоматической фокусировки можно с помощью следующей функции, представленной в листинге 2.24.

Листинг 2.24. Функция, проверяющая поддержку автоматической фокусировки на элементах управления Web-формы

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

Вместо самостоятельного написания этой функции, можно воспользоваться библиотекой `Modernizr` (версии 1.1 или более новой), чтобы выявить под-

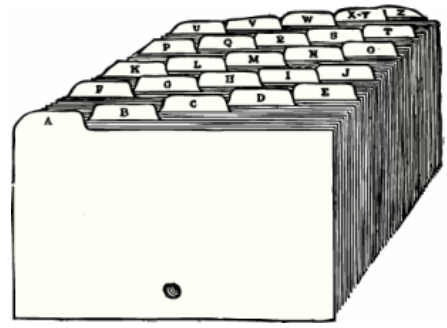
держку автоматической фокусировки на полях Web-формы, как показано в листинге 2.25.

Листинг 2.25. Проверка наличия поддержки функции autofocus с помощью Modernizr

```
if (Modernizr.input.autofocus) {
  // Автофокусировка работает!
} else {
  // автофокусировка не поддерживается :(
  // нужно прибегнуть к решению на базе скрипта
}
```

Микроданные (Microdata)

Микроданные (Microdata) — это стандартизованный метод добавления на ваши Web-страницы различной информации (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#microdata**). Например, вы можете использовать микроданные, чтобы объявить о том, что фотография доступна по конкретной лицензии Creative Commons (**http://creativecommons.org/choose/**). Как вы увидите в дальнейшем в *главе 10*, посвященной распределенным расширениям, вы можете использовать микроданные, например, чтобы разметить конкретную Web-страницу (такую, как "About Me"). Браузеры, браузерные расширения, а также поисковики могут преобразовывать ваши микроданные HTML5 в формат vCard (**https://secure.wikimedia.org/wikipedia/en/wiki/VCard**), стандартный формат для представления контактной информации. Кроме того, вы можете определять собственные словари микроданных.



Стандартные микроданные HTML5 включают как разметку HTML (главным образом, для поисковых машин), так и набор DOM-функций (в основном, для браузеров). Разметку микроданных можно включать в Web-страницы. Она представляет собой не что иное, как несколько удачно размещенных атрибутов, и поисковики, которые не могут интерпретировать атрибуты микроданных, просто будут их игнорировать. Но если вам требуется получать доступ к микроданным и манипулировать ими через DOM, вам нужно проверить, поддерживает ли ваш браузер microdata DOM API.

Проверка поддержки API микроданных в формате HTML5 использует методику определения поддержки номер 1 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает API микроданных HTML5, то глобальный объект документа будет иметь функцию `getItems()`. В противном случае функция `getItems()` не будет определена. Функция, позволяющая выявить поддержку микроданных HTML5, представлена в листинге 2.26.

Листинг 2.26. Функция, позволяющая определить поддержку микроданных HTML5

```
function supports_microdata_api() {
  return !!document.getItems;
}
```

`Modernizr` еще не поддерживает API микроданных, поэтому вам необходимо написать функцию самостоятельно, так, как только что было продемонстрировано.

History API

API истории HTML5 (**http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html**) представляет собой стандартизированный способ манипулирования историей посещений Web-страниц в браузере через скрипты. Часть этого API — навигация по ссылкам — была доступна и в предшествующих версиях HTML. Новая часть, введенная в HTML5, представляет новый способ добавления записей в историю навигации браузера и реагирования на их добавления и удаления в стек, когда пользователь нажимает кнопку **Back**. Это значит, что URL могут продолжать выполнять свою работу в качестве уникальных идентификаторов текущего ресурса, даже если приложения, иненсивно использующие скрипты, не выполняют полного обновления страницы.

Проверка поддержки HTML5 history API подразумевает использование техники детекции номер 1 (см. *разд. "Методика выявления поддержки функций HTML5"*). Если ваш браузер поддерживает HTML5 history API, то в глобальном объекте `history` будет присутствовать функция `pushState()`. Если ваш браузер не поддерживает history API, то функция `pushState()` будет не определена (листинг 2.27).

Листинг 2.27. Функция `supports_history_api()` для поддержки браузером HTML5 history API

```
function supports_history_api() {
  return !!(window.history && history.pushState);
}
```

Вместо самостоятельного написания этой функции вы можете использовать библиотеку `Modernizr` (версии 1.6 или более новой) для выявления поддержки HTML5 history API, как показано в листинге 2.28.

Листинг 2.28. Проверка поддержки history API с помощью `Modernizr`

```
if (Modernizr.history) {
  // возможно манипулирование историей!
} else {
  // манипулирование историей не поддерживается :(
  // Можно попробовать воспользоваться решением
  // на базе скриптов наподобие History.js
  // (https://github.com/balupton/History.js/)
}
```

Материалы, рекомендуемые для дальнейшего чтения

Спецификации и стандарты:

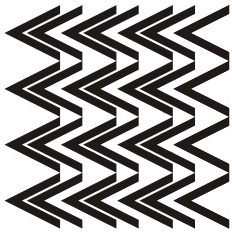
- Элемент `<canvas>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html**
- Элемент `<video>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#video**
- Типы `<input>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#states-of-the-type-attribute**
- Атрибут `<input placeholder>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/common-input-element-attributes.html#the-placeholder-attribute**
- Атрибут `<input autofocus>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#autofocusing-a-form-control**
- Хранилище HTML5 — **http://dev.w3.org/html5/webstorage/**
- Web Workers — **http://www.whatwg.org/specs/web-workers/current-work/**
- Автономные Web-приложения — **http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html#offline**
- Geolocation API — **http://www.w3.org/TR/geolocation-API/**
- Раздел *"Session History and Navigation"* из черновика стандарта HTML5 — **http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html**

Библиотеки JavaScript:

- `Modernizr`, библиотека для выявления поддержки HTML5 (**http://www.modernizr.com/**)
- `geo.js`, "обертка" для geolocation API (**https://code.google.com/p/geo-location-javascript/**)

Другие статьи и учебные курсы:

- Video for Everybody! — **http://camendesign.com/code/video_for_everybody**
- Статья *"A gentle introduction to video encoding"* — **http://diveintomark.org/tag/give**
- Статья *"Video type parameters"* — **http://wiki.whatwg.org/wiki/Video_type_parameters**
- Приложения 1 и 2 к этой книге



Глава 3

Что все это означает?

В данной главе мы возьмем в качестве примера HTML-страницу, не содержащую никаких ошибок, и усовершенствуем ее. Некоторые из частей нашей Web-страницы станут короче, но некоторые, наоборот, станут длиннее. Однако в целом наша Web-страница станет более понятной и удобочитаемой. Окончательный результат должен вам понравиться!

Вот страница, которую мы будем модернизировать: **http://diveintohtml5.org/examples/blog-original.html**. Ознакомьтесь с ней, изучите и попробуйте ее даже полюбить. Затем откройте страницу на новой вкладке и не возвращайтесь обратно, по крайней мере, до тех пор, пока вам не захочется выбрать в браузере команду **View Source** (Исходный код страницы) хотя бы один раз.

Объявление типа документа (Doctype)

В верхней строке вы увидите следующее содержимое, приведенное в листинге 3.1.

Листинг 3.1. Первая строка страницы, изучаемой в качестве примера

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

Этот элемент называется "объявлением типа документа", "типом документа" или просто элементом (тегом) `doctype`. Он имеет довольно длительную историю и основывается на том, что раньше считалось "черной магией". В течение своей работы над версией Internet Explorer 5 для Mac, разработчики Microsoft столкнулись с удивительной проблемой. Новая, еще только планируемая к выпуску версия их браузера повысила свои уровни соответствия стандартам настолько, что более старые страницы уже не могли отображаться надлежащим образом. Точнее говоря, они отображались надлежащим образом (согласно спецификациям), но проблема заключалась в том, что пользователям этого как раз и не требовалось. Наоборот, пользователи ожидали, что страница будет отображаться не в соответствии со спецификациями, а так, как они, пользователи, привыкли. Страницы как таковые создавались с учетом "хитростей" браузеров, которые доминировали на рынке

на момент создания тех самых страниц, причем в основном это были Netscape 4 и Internet Explorer 4. Браузер IE5/Mac оказался настолько "продвинутым", что он, фактически, устроил в Web настоящую революцию!

Тогда корпорация Microsoft выступила с инновационным решением. Перед визуализацией страницы браузер IE5/Mac искал элемент `doctype`, который обычно представлял собой первую строку исходного кода HTML (предшествующую даже элементу `<html>`). Более старые страницы, авторы которых полагались на особенности более старых браузеров, обычно вообще не имели элемента `doctype`. IE5/Mac визуализировал эти страницы точно так же, как это делали более старые браузеры. Чтобы "активировать" поддержку новых стандартов, авторы Web-страницы должны были явным образом указать на это, снабдив свои Web-страницы элементом `doctype`, предшествующим элементу `<html>`.

Эта идея начала распространяться со скоростью лесного пожара, и очень скоро все ведущие браузеры имели два режима: *режим поддержки старых "особенностей"* (quirks mode) и *режим поддержки новых стандартов* (standards mode). Естественно, поскольку все это происходило в Web, контроль над ситуацией вскоре был утрачен. Когда разработчики Mozilla попытались выпустить версию 1.1 своего браузера, они обнаружили, что некоторые страницы визуализировались в режиме поддержки стандартов, в то время как на самом деле они полагались на специфические "особенности". Разработчики Mozilla буквально только что модернизировали свой движок, отвечающий за отображение, и тысячи страниц в одночасье перестали отображаться "нормально" (т. е. так, как того ожидали их авторы). Тогда появился — и это не моя выдумка — "практически стандартный режим" или Almost Standards Mode (**https://developer.mozilla.org/en/Gecko%27s_%22Almost_Standards%22_Mode**).

В своей семестровой работе, озаглавленной *"Activating Browser Modes with Doctype"* (Активация режимов браузера с помощью `Doctype`)[1], Генри Сайвонен (Henri Sivonen) обобщил информацию о различных режимах визуализации HTML-содержимого браузерами следующим образом:

- *Режим поддержки "особенностей"* (Quirks Mode)

  В режиме поддержки "особенностей" (Quirks mode) браузеры нарушают спецификации современного формата Web, чтобы избежать отказов в отображении страниц, созданных в соответствии со старыми соглашениями, которые доминировали до конца 1990-х.

- *Режим поддержки стандартов* (Standards Mode)

  В режиме поддержки стандартов браузеры делают попытку отобразить документы, соответствующие новым стандартам, в соответствии с требованиями, установленными этими стандартами. Делают они это до известной степени, диктуемой особенностями конкретного браузера. В стандарте HTML5 этот режим называется "без особенностей" (no quirks mode).

- *Режим жесткого соответствия стандартам* (Almost Standards Mode)

  Firefox, Safari, Chrome, Opera (начиная с версии 7.5) и Internet Explorer 8 имеют режим "практически полного соответствия стандартам" (Almost Standards

[1] Оригинал этой работы см. здесь: **http://hsivonen.iki.fi/doctype/**.

Mode), который реализует вертикальное масштабирование ячеек таблиц традиционно и не обязательно жестко (traditionally and not rigorously) в соответствии со спецификацией CSS2. В HTML5 этот режим называется *режимом ограниченной поддержки особенностей* (limited quirks mode).

Примечание

Вам следует прочесть и остальные части ранее упомянутой статьи Генри, поскольку мое изложение здесь является сильно упрощенным. Даже в IE5/Mac имелось несколько старых типов документов (doctypes), которые не принимались в расчет, когда речь заходила о поддержке стандартов. Со временем список "особенностей" рос, и то же самое происходило и со списком типов документов (doctypes), которые "запускали" режим поддержки "особенностей (quirks mode). Когда я в последний раз попытался выполнить подсчет, имелось 5 типов документов (doctypes), которые активировали режим практически полного соответствия стандартам, и целых 73 типа, которые активировали режим "поддержки особенностей" (quirks mode). Но вполне вероятно, что я что-то и упустил из виду, так что я даже не собираюсь заострять здесь внимание на такой абсурдной чепухе, как то, что Internet Explorer 8 переключается между целыми четырьмя (!) различными режимами визуализации. Вот блок-схема, иллюстрирующая весь процесс: **http://hsivonen.iki.fi/doctype/ie8-mode.png**. Эту чушь надо выжигать каленым железом!

Итак, к чему мы пришли? Ах, да, мы обсуждаем типы документов (doctype). Давайте еще раз посмотрим на объявление типа документа, которое было представлено в листинге 3.1.

Листинг 3.1

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

Обычно это оказывается один из 15 типов документа, запускающих режим соответствия стандартам (standards mode) для всех современных браузеров. Здесь нет ничего "криминального". Если вам это нравится, вы вполне можете продолжать этого и придерживаться. Или же вы можете изменить тип документа на тип документа HTML5, что намного проще, короче и тоже запускает режим соответствия стандартам.

Объявление типа документа (`doctype`) в HTML5 выглядит так:

```
<!DOCTYPE html>
```

И это все. Всего 15 символов. Это настолько просто, что вы можете ввести этот код вручную и больше ни о чем не беспокоиться.

Элемент Root

HTML-страница представляет собой последовательность вложенных элементов. Вся структура страницы построена по древовидной иерархии. Некоторые элементы представляют собой "отростки", как, например, две ветви этого дерева, отходящие

от основного "ствола". Некоторые элементы являются "потомками" других элементов, как небольшая ветвь, отходящая от другой, более мощной ветви. Справедливо и обратное. Элемент, который содержит в своем составе другие элементы, считается "родительским" узлом по отношению ко всем своим непосредственным "потомкам" и "предком" всех их "потомков". Элементы, которые не имеют потомков, называются "листьями" ("leaf" nodes). Самый верхний элемент в этой иерархии, являющийся предком всех остальных элементов, называется "корнем" или корневым элементом (root element). Корневым элементом HTML-страницы всегда является элемент `<html>`.

В этой странице (**http://diveintohtml5.org/examples/blog-original.html**) корневой элемент выглядит так, как показано в листинге 3.2.

Листинг 3.2. Корневой элемент Web-страницы, изучаемой в нашем примере

```
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="en"
      xml:lang="en">
```

С этой разметкой все в порядке, здесь нет никаких ошибок. Опять же, если вам это нравится, вы можете оставить все "как есть". Это вполне легальный код HTML5. Однако некоторые фрагменты этого кода в HTML5 уже не являются обязательными, поэтому вы вполне можете сэкономить несколько байт, удалив их.

Первый из элементов, которые мы обсудим — это атрибут `xmlns`. Это — наследие XHTML 1.0 (**http://www.w3.org/TR/xhtml1/**). Он указывает на то, что элементы на данной странице принадлежат к пространству имен XHTML, **http://www.w3.org/1999/xhtml**. Но элементы страниц, соответствующих стандарту HTML5, всегда находятся в этом пространстве имен (см. **http://www.whatwg.org/specs/web-apps/current-work/multipage/infrastructure.html#xml**), так что вам больше нет необходимости декларировать это явным образом. Ваша Web-страница в формате HTML5 будет работать абсолютно одинаково во всех браузерах, вне зависимости от того, присутствует данный атрибут или нет.

Пропуск атрибута `xmlns` оставляет нам корневой элемент Web-страницы следующего вида:

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

Здесь присутствуют два атрибута, `lang` и `xml:lang`, оба из которых определяют язык данной HTML-страницы. Например, значение `en` указывает на то, что данная конкретная страница написана на английском языке (English). Если вы хотите

писать свою Web-страницу на другом языке, то найдите код нужного вам языка[1]. Зачем нужны целых два атрибута для того, чтобы указать на то же самое? Опять же, это — наследие XHTML. Только атрибут `lang` имеет значение в коде, соответствующем стандарту HTML5. Если вам хочется, вы можете сохранить атрибут `xml:lang`, но, если вы хотите это сделать, вам нужно гарантировать, что он задает тот же самый язык, что и атрибут `lang`[2]. В стандарте говорится буквально следующее:

> "Чтобы упростить переход к XHTML и обратно, авторы могут указать атрибут, не принадлежащий ни одному из пространств имен, не имеющий префикса и с локальным именем в виде строки литералов `xml:lang` для элементов HTML в HTML-документах, но такие атрибуты должны указываться, только если указан и атрибут `lang`, не принадлежащий ни одному из пространств имен, причем оба атрибута должны иметь одно и то же значение в кодировке ASCII без учета регистра символов клавиатуры. Атрибут, не принадлежащий ни к одному из пространств имен, не имеющий префикса и имеющий только локальное имя "`xml:lang`", не оказывает влияния на обработку языка".

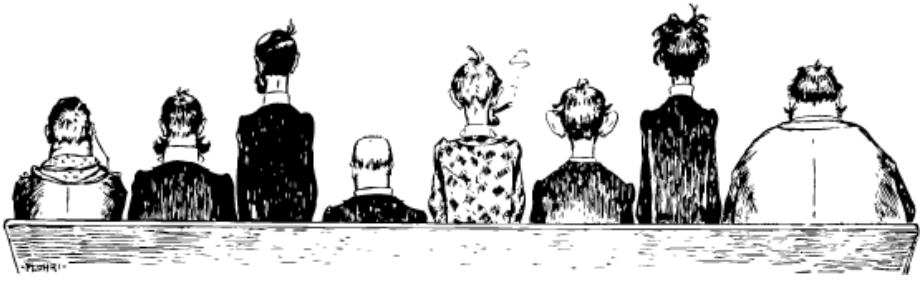
Вы готовы от него отказаться? Это нормально, пусть так и будет. Отказываемся, отказываемся... Отказались! В результате мы остаемся с корневым элементом следующего вида:

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

И это все, что я хотел сказать по данному поводу.

Элемент <head>

Обычно первым дочерним элементом корневого элемента является элемент `<head>`. Элемент `<head>` содержит метаданные — информацию о странице, а не ее тело. Тело самой страницы, что совсем не удивительно, заключается в элементе `<body>`. Сам по себе элемент `<head>` достаточно скучен, и в HTML5 в него не было внесено ничего нового или интересного. Все, что представляет интерес, заключено *внутри* элемента `<head>`. Чтобы рассмотреть все эти вещи, вернемся к нашей странице с примером. Интересующий нас фрагмент кода этой страницы приведен в листинге 3.3



[1] О том, как сделать это грамотно, прочтите здесь: **http://www.w3.org/International/questions/qa-choosing-language-tags**.

[2] См. **http://www.whatwg.org/specs/web-apps/current-work/multipage/elements.html#the-lang-and-xml:lang-attributes**.

Листинг 3.3. Пример элемента <head>

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>My Weblog</title>
  <link rel="stylesheet" type="text/css" href="style-original.css" />
  <link rel="alternate" type="application/atom+xml"
                        title="My Weblog feed"
                        href="/feed/" />
  <link rel="search" type="application/opensearchdescription+xml"
                        title="My Weblog search"
                        href="opensearch.xml"  />
  <link rel="shortcut icon" href="/favicon.ico" />
</head>
```

Начнем с рассмотрения элемента `<meta>`.

Кодировка символов

Когда речь заходит о "тексте", вы, вероятно, ожидаете, что мы начнем обсуждать "те символы, которые видим на экране". Но в действительности компьютеры имеют дело не с буквами и символами, а с битами и байтами. Любой фрагмент текста, который вы видите на экране, на самом деле представляет собой поток битов в определенной символьной кодировке. Существуют сотни различных символьных кодировок (**http://www.iana.org/assignments/character-sets**), часть которых оптимизирована для передачи текста на конкретных языках (например, русском, английском или китайском), в то время как другие кодировки могут использоваться для передачи текста на множестве языков. Обобщенно говоря, кодировка символов представляет собой установку соответствия между буквами и символами, которые вы видите на экране, и той информацией, которая фактически загружается в память вашего компьютера и хранится на его жестком диске.

В действительности же все еще сложнее, чем это обобщенное описание. Один и тот же символ может появляться во множестве различных кодировок, но каждая кодировка для представления такого символа в памяти или на диске должна использовать свою последовательность битов. Таким образом, символьную кодировку можно рассматривать как разновидность ключа для декодирования текста. Каждый раз, когда кто-либо передает вам последовательность битов и сообщает о том, что она представляет собой текст, вам необходимо знать, какая кодировка использовалась для этого текста. Зная кодировку, вы сможете декодировать последовательность битов в символы и отобразить их на экране (или обработать их тем либо иным способом).

Тогда возникает вопрос: каким образом браузер определяет кодировку текста, передаваемого через Web в виде потока битов? Если у вас возник этот вопрос,

вы находитесь на правильном пути. Если вы знакомы с заголовками HTTP, то возможно, вам встречались заголовки наподобие следующего:

```
Content-Type: text/html; charset="utf-8"
```

Вкратце говоря, этот заголовок сообщает о том, что Web-сервер "считает", что отсылает вам HTML-документ и что этот документ использует кодировку символов UTF-8. К сожалению, во всем этом "бульоне", который представляет собой World Wide Web, лишь немногие авторы действительно могут иметь контроль над своим HTTP-сервером. Возьмем, к примеру, Blogger (**http://www.blogger.com**): информационное наполнение создается отдельными личностями, но управление сервером осуществляет компания Google. Поэтому стандарт HTML 4 и предоставил способ указания кодировки в документе HTML как таковом. Возможно, вам встречались и такие примеры:

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

Вкратце, такой заголовок указывает, что разработчики Web-страницы считают, что они создали HTML-документ, используя кодировку UTF-8.

Оба упомянутых подхода продолжают работать и в HTML5. Заголовок `http` — это предпочтительный метод, и он имеет приоритет перед тегом `<meta>`, если он вообще присутствует. Но не каждый пользователь может определять заголовки HTTP, так что тег `<meta>` все еще не вышел из употребления. В действительности же, в HTML5 все существенно упростилось. Теперь заголовок выглядит так:

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

Это будет работать во всех браузерах. Как это сократило и упростило синтаксис? Вот лучшее объяснение, которое мне удалось найти: **http://lists.w3.org/Archives/Public/public-html/2007Jul/0550.html**.

> Объяснение комбинированного атрибута `<meta charset="">` заключается в том, что пользовательские агенты (UA) уже реализуют эту комбинацию, потому что большинству людей свойственно оставлять цитаты без кавычек, например:
>
> ```
> <META HTTP-EQUIV=Content-Type CONTENT=text/html; charset=ISO-8859-1>
> ```

Существуют еще несколько тестовых примеров `<meta charset>`, которые докажут вам (если вы все еще не верите), что браузеры эту функцию уже реализуют: **http://simon.html5.org/test/html/parsing/encoding/**.

Вопрос к Профессору Разметки

В: Я никогда не использую "смешных" символов. Мне по-прежнему необходимо декларировать кодировку?

О: Да! Вы *всегда* должны декларировать кодировку для каждой обслуживаемой вами HTML-страницы. Если вы не укажете кодировку, это может привести к возникновению уязвимостей в системе безопасности (см., например: **https://code.google.com/p/doctype/wiki/ArticleUtf7**).

Подводя итоги, можно сказать, что кодировка символов — это довольно сложный вопрос, который с течением времени проще не стал, особенно с учетом плохо написанного программного обеспечения, авторы которого, пожалуй, не умели

и не умеют ничего другого, кроме копирования и вставки. Вам следует *всегда* указывать кодировку символов для *любого* HTML-документа, так как в противном случае вам не избежать неприятностей. Сделать это можно с помощью заголовка HTTP `Content-Type`, декларации `<meta http-equiv>` или с помощью еще более краткой декларации `<meta charset>`, но сделать это необходимо обязательно. Web отблагодарит вас за это.

Ссылочные отношения

Обычные ссылки (`<a href>`) просто указывают на другую страницу. Отношения ссылок предоставляют способ объяснить, *почему* вы указываете на другую страницу. Эти слова дают один из следующих вариантов завершения предложения "я ссылаюсь на другую страницу, потому что ..."

- ...она представляет собой таблицу стилей, которая содержит правила CSS, которые ваш браузер должен применить к этому документу;
- ...это канал (feed), который содержит ту же информацию, что и данная страница, но в стандартном формате, на который возможна подписка;
- ...это перевод данной страницы на другой язык;
- ...она содержит ту же информацию, что и данная страница, но в формате PDF;
- ...это следующая глава онлайновой книги, частью которой является и данная страница.

И так далее. HTML5 подразделяет отношения ссылок на две категории (**http://www.whatwg.org/specs/web-apps/current-work/multipage/semantics.html#attr-link-rel**):

> "С помощью элементов-ссылок могут создаваться две категории ссылок. Ссылки на внешние ресурсы (links to external resources) — это ссылки на те ресурсы, которые должны использоваться для дополнения текущего документа, а гиперссылки (hyperlinks) представляют собой ссылки на внешние документы. ...
>
> Точное поведение ссылок на внешние ресурсы зависит от точного взаимоотношения, определенного для соответствующего типа ссылки".

В примерах, только что приведенных мною, только первая ссылка (`rel="stylesheet"`) представляет собой ссылку, ведущую к внешнему ресурсу. Все остальные ссылки представляют собой гиперссылки, ведущие к другим документам. Вы можете следовать по этим ссылкам, а можете этого и не делать. В любом случае, они не являются необходимыми для просмотра текущей страницы.

Чаще всего, ссылочные отношения встречаются для элементов `<link>`, находящихся в пределах элемента `<head>` текущей страницы. Некоторые ссылочные связи могут применяться для элементов `<a>`, но, даже при том, что это допустимо, такие отношения встречаются нечасто. Кроме того, HTML5 позволяет устанавливать отношения для элементов `<area>`, но это встречается еще реже. (В HTML 4 не допускалась установка атрибута `rel` на элементы `<area>`.) Чтобы проверить, где можно использовать конкретные значения `rel`, см. полную схему отношений между ссылками (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#linkTypes**).

Вопрос к Профессору Разметки

В: Могу ли я определять собственные ссылочные отношения?

О: По всей видимости, запас идей новых видов ссылочных отношений неисчерпаем. В попытке помешать тому, чтобы создание таких отношений стало самоцелью (**http://developer.apple.com/library/safari/#documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html**), WHATWG поддерживает реестр предполагаемых значений атрибута `rel` (**http://wiki.whatwg.org/wiki/RelExtensions**) и определяет процедуру их принятия (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#other-link-types**).

rel = stylesheet

Рассмотрим первое ссылочное отношение в нашем примере:

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

Это — наиболее часто встречающееся ссылочное отношение в мире (в самом буквальном смысле). Отношение `<link rel="stylesheet">` используется для связывания страницы с правилами CSS, хранящимися в отдельном файле. Небольшая оптимизация, которой вы можете воспользоваться в HTML5, заключается в пропуске атрибута `type`. В Web существует только один язык таблиц стилей, CSS, поэтому его название и используется в качестве значения по умолчанию для атрибута `type`. Это правило работает во всех браузерах. Я предполагаю, что, может быть, кто-нибудь со временем и изобретет новый язык таблиц стилей, но, даже если это и произойдет, вам достаточно будет вернуться к использованию атрибута `type`.

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

rel = alternate

Продолжим обсуждение нашего примера страницы. В листинге 3.3 имелся следующий фрагмент:

```
<link rel="alternate"
      type="application/atom+xml"
      title="My Weblog feed"
      href="/feed/" />
```

Это ссылочное отношение тоже является довольно распространенным. Отношение `<link rel="alternate">`, в комбинации с типами RSS или Atom, указанными для атрибута `type`, позволяет реализовать возможность, известную как "автоматическое распознавание каналов" (feed autodiscovery). Это позволяет приложениям для чтения потоков синдикации (наподобие Google Reader) определять, имеются ли на сайте анонсы заголовков новых статей, распространяемых по подписке. Большинство браузеров поддерживают автоматическое распознавание потоков и отображают специальный значок справа от URL. В отличие от ситуации с `rel="stylesheet"`, атрибут `type` здесь имеет значение, поэтому пропускать его нельзя!

Ссылочное отношение `rel="alternate"` всегда представляло собой странный гибрид вариантов использования, даже в HTML 4 (**http://www.w3.org/TR/html401/**

types.html#type-links). В HTML5, его определение было уточнено и расширено таким образом, чтобы более точно описывать существующее Web-содержимое. Как вы только что видели, использование `rel="alternate"` в сочетании с `type=application/atom+xml` указывает на то, что на текущей странице имеется канал Atom. Кроме того, вы можете использовать `rel="alternate"` в сочетании с другими атрибутами типа, чтобы указать на возможность получения в точности той же информации в других форматах, например, в таких, как PDF.

HTML5 также кладет конец долго существовавшей путанице с тем, как связывать документы и их переводы. HTML 4 предписывает использовать атрибут `lang` в сочетании с `rel="alternate"`, чтобы указать язык связанного документа, но это неправильно. Документ *HTML 4 Errata* (**http://www.w3.org/MarkUp/html4-updates/errata**) перечисляет четыре наиболее явных ошибки в спецификации HTML 4. Одной из этих ошибок является способ указания языка связанного документа, с помощью `rel="alternate"`. Правильный способ, описанный ранее в документе HTML 4 Errata, а теперь и в HTML5, заключается в использовании атрибута `hreflang`. К сожалению, этот документ со списком ошибок и с дополнениями никогда не был включен в спецификацию HTML 4, потому что к тому моменту никто в рабочей группе W3C больше не работал над HTML.

Другие ссылочные отношения в HTML5

Атрибут `rel="archives"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#rel-archives**) указывает, что документ, на который производится ссылка, описывает коллекцию записей, документов или других материалов, представляющих исторический интерес. Индексная страница блога может ссылаться на индексы более старых записей с помощью атрибута `rel="archives"`.

Атрибут `rel="author"` используется для ссылок на информацию об авторе страницы. Это может быть ссылка наподобие `mailto: address`, хотя это и не обязательно должно быть так. Это может быть просто ссылка на форму с контактной информацией или на страницу "About the author".

Атрибут `rel="external"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-external**) указывает на то, что ссылка ведет к документу, который не является частью сайта, к которому принадлежит текущий документ. Я считаю, что впервые этот атрибут был популяризован WordPress (**http://wordpress.org/**), так как этот сайт применяет его со ссылками, оставляемыми комментаторами.

HTML 4 определяет атрибуты `rel="start"`, `rel="prev"` и `rel="next"` (**http://www.w3.org/TR/html401/types.html#type-links**) для определения отношений между страницами, которые являются частью некоторой последовательности

(например, такой, как следующие одна за другой главы в книге, или же серия последовательных записей в блоге). Единственный из них, который когда-либо использовался правильно — это атрибут `rel="next"`. Люди использовали `rel="previous"` вместо `rel="prev"`; они использовали атрибуты `rel="begin"` и `rel="first"` вместо `rel="start"`; они писали `rel="end"` вместо `rel="last"`. Ну и, наконец, все пользовались атрибутом `rel="up"`, чтобы сослаться на "родительскую" страницу.

HTML5 включает атрибут `rel="first"`, который представлял собой наиболее общую вариацию одного из различных способов сказать "первая страница в последовательности" (`rel="start"` — это нестандартный синоним, используемый для обеспечения обратной совместимости). Кроме того, HTML5 также включает атрибуты `rel="prev"` и `rel="next"`, как и HTML 4, и поддерживает атрибут `rel="previous"` для обеспечения обратной совместимости, как и `rel="last"` (последний документ в последовательности, противоположность `rel="first"`) и `rel="up"`.

Наилучший способ представить себе цель атрибута `rel="up"` — посмотреть на свою панель иерархической навигации (breadcrumb navigation) или, как минимум, представить ее себе. Ваша домашняя страница, вероятно, является первой в списке, а текущая страница находится в самом конце. Атрибут `rel="up"` указывает на предпоследнюю страницу в списке посещенных.

Атрибут `rel="icon"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#rel-icon**) — это второй наиболее популярный атрибут, указывающий на отношение ссылок между страницами, после атрибута `rel="stylesheet"` (**http://code.google.com/intl/ru-RU/webstats/2005-12/linkrels.html**). Обычно этот атрибут указывается вместе с ярлыком, например:

```
<link rel="shortcut icon" href="/favicon.ico">
```

Все основные браузеры поддерживают этот вариант, чтобы ассоциировать со страницей небольшой значок. Обычно он отображается в адресной строке браузера рядом с URL, или на вкладке браузера, или и там и там.

Еще одно новшество HTML5: атрибут `sizes` может использоваться в сочетании со значком, чтобы указывать размер значка, на который дается ссылка (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#rel-icon**).

Атрибут `rel="license"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-license**) был придуман сообществом микроформатов (**http://microformats.org/wiki/rel-license**). Он "указывает, что документ, на который дается ссылка, содержит лицензионные условия и информацию об авторских правах, на условиях которых этот документ предоставляется".

Атрибут `rel="nofollow"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-nofollow**) "указывает, что ссылка не была предоставлена (endorsed) изначальным автором или публикатором страницы или что ссылка на документ включена, главным образом, потому, что между авторами или лицами, опубликовавшими две страницы, существуют коммерческие взаимоотношения". Этот атрибут был изобретен компанией Google (**http://googleblog.blogspot.com/2005/01/preventing-comment-spam.html**) и стандартизован сообществом микро-

форматов (**http://microformats.org/wiki/rel-nofollow**). Сайт WordPress добавляет атрибут `rel="nofollow"` к ссылкам, добавляемым комментаторами. При этом имелось в виду, что если ссылки `"nofollow"`не отражаются на рейтинге страницы (PageRank), то спамеры бросят попытки помещать в блоги мусорные комментарии. Этого не произошло, но использование `rel="nofollow"` вошло в обиход.

Атрибут `rel="noreferrer"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-noreferrer**) "указывает, что при прохождении по ссылке не будет происходить утечки информации о реферререре (referrer)". Ни один из поставляющихся на текущий момент браузер этого не поддерживает, но недавно поддержка была добавлена в ночные сборки WebKit (**http://www.webkit.org/blog/907/webkit-nightlies-support-html5-noreferrer-link-relation/**), так что со временем она должна появиться и в Safari, Google Chrome, а также других браузерах, основывающихся на WebKit. Тестовую страницу для `rel="noreferrer"` можно найти здесь: **http://wearehugh.com/public/2009/04/rel-noreferrer.html**.

`rel="pingback"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-pingback**) указывает адрес "pingback"-сервера. Как объясняется в спецификации Pingback (**http://hixie.ch/specs/pingback/pingback-1.0**), "система обратной отсылки команды **ping** (pingback system) предоставляет способ, с помощью которого Web-журнал (блог) может автоматически ставиться в известность о том, когда другие Web-сайты ссылаются на него. ... Это предоставляет и обратную возможность — обратные ссылки, позволяющие пройти обратно по цепочке ссылок". Системы ведения блогов, в частности, WordPress, реализуют механизм pingback для уведомления авторов, на блоги которых вы поставили ссылки при написании новой записи в своем Web-журнале.

Атрибут `rel="prefetch"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-prefetch**) "указывает, что выборка и кэширование с упреждающим вытеснением, вероятно, окажет благоприятное влияние, потому что, вероятнее всего, пользователю потребуется этот ресурс". Иногда поисковые системы добавляют на страницу результатов поиска следующие ссылки `<link rel="prefetch" href="URL of top search result">`, если есть основания считать, что первые в списке результаты намного популярнее остальных. Например: воспользуйтесь Firefox, выполните в Google поиск CNN, просмотрите исходный код страницы, а затем выполните поиск по ключевому слову `prefetch`. Mozilla Firefox — это, на сегодняшний день, единственный браузер, который поддерживает атрибут `rel="prefetch"`.

Атрибут `rel="search"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-search**) "указывает, что документ, на который дается ссылка, предоставляет интерфейс специально для поиска документа и относящихся к нему ресурсов". В частности, если вы хотите воспользоваться атрибутом

`rel="search"`, чтобы сделать что-нибудь полезное, он должен указывать на документ OpenSearch (**http://www.opensearch.org/Home**), который описывает, как браузер может сконструировать URL для поиска по заданному ключевому слову на текущем сайте. OpenSearch (и ссылки `rel="search"`, указывающие на документы описаний OpenSearch) поддерживаются в Microsoft Internet Explorer, начиная с версии 7, а в Mozilla Firefox — начиная с версии 2.

Атрибут `rel="sidebar"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-sidebar**) "указывает, что документ, на который дается ссылка, в случае его извлечения, предназначен для того, чтобы отображаться (по возможности) во вторичном контексте просмотра, а не в текущем контексте браузера". Что это означает? В таких браузерах, как Opera и Mozilla Firefox, это означает, что при щелчке мышью по ссылке, нужно предложить пользователю создать закладку, которая, будучи выбрана из меню **Bookmarks**, откроет связанный документ в боковой панели браузера.

Примечание

В Opera эта область называется "панелью" (panel).

Internet Explorer, Safari и Google Chrome игнорируют атрибут `rel="sidebar"` и обращаются со ссылкой как с обычной ссылкой. Тестовую страницу, демонстрирующую использование атрибута `rel="sidebar"` можно найти здесь: **http://wearehugh.com/public/2009/04/rel-sidebar.html**.

`rel="tag"` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-tag**) "указывает, что тег, представляемый документом, на который дается ссылка, применим к текущему документу". Маркировка тегов (tags), представляющих собой ключевые слова, обозначающие категории, атрибутом `rel`, была изобретена сообществом Technorati (**http://www.powazek.com/2005/07/000532.html**), чтобы помочь членам сообщества классифицировать записи в блогах. В ранних блогах и учебных пособиях по пользованию ими эти теги так и назывались — "тегами Technorati". (Да-да, вы читаете все правильно: коммерческая фирма сумела убедить весь мир добавить в Web мета-данные, чтобы помочь этой компании выполнять ее работу. Замечательная работа в своем роде, если вы можете это оценить!) Синтаксис впоследствии был стандартизован в сообществе микроформатов (**http://microformats.org/wiki/rel-tag**), где он свелся к простому `rel="tag"`. Большинство систем ведения блогов, которые позволяют ассоциировать с записями теги, категории, ключевые слова, маркируют их ссылками `rel="tag"`. Браузеры не делают с ними ничего особенного; они разработаны специально для поисковиков, чтобы пояснять, какой теме посвящена та или иная страница.

Новые семантические элементы HTML5

Предназначение HTML5 заключается не только в том, чтобы сделать существующую разметку короче и лаконичнее (несмотря на то, что в значительной мере новый стандарт делает именно это). Помимо этого, HTML5 определяет и новые семантические элементы, перечисленные в табл. 3.1.

Таблица 3.1. *Новые семантические элементы HTML5*

Элемент	Описание
<section>	Элемент section (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-section-element**) представляет раздел типового документа или приложения. В данном контексте раздел (section) представляет собой тематическую группировку информационного содержимого, как правило, снабженную заголовком. В качестве примеров разделов можно привести главы, страницы на вкладках диалогового окна, предоставляющего возможность просмотра на вкладках, или же нумерованные разделы диссертации. Домашняя страница Web-сайта может быть разбита на такие разделы, как введение, новости, контактная информация
<nav>	Элемент nav (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-nav-element**) представляет собой раздел страницы, который ссылается на другие страницы или на другие части в пределах той же страницы: это навигационный раздел. Не все группы ссылок на странице должны представлять собой элементы nav — только разделы, состоящие из основных навигационных блоков, подходят для использования в качестве элементов nav. В частности, нижние колонтитулы обычно содержат короткий список ссылок на общие страницы сайта, такие, как условия обслуживания, домашняя страница, страница с соглашением об авторских правах и т. д. Элемент нижнего колонтитула (footer) сам по себе уже достаточен для таких случаев, и никакой необходимости в элементах nav нет
<article>	Элемент article (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-article-element**) представляет компонент страницы, состоящий из самодостаточной композиции в документе, на странице, в приложении, на сайте, и предназначается для независимого распространения и многократного использования, например, в синдикации (syndication). Это может быть, например, сообщение на форум, статья в газете или журнале, запись в блоге, комментарий, присланный пользователем, интерактивный виджет (widget) или гаджет (gadget), или же любой другой независимый элемент или информационное наполнение
<aside>	Элемент aside (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-aside-element**) представляет собой раздел страницы, который состоит из информационного наполнения, которое имеет косвенное отношение к контенту, окружающему этот элемент, и которое может рассматриваться отдельно от него. Такие разделы в печатном виде часто представляются в виде полей или боковых панелей. Элемент может применяться для создания различных типографских эффектов наподобие врезок или боковых панелей, для групп элементов nav, а также для другого информационного наполнения, которое считается отдельным от основного информационного наполнения страницы
<hgroup>	Элемент hgroup (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-hgroup-element**) представляет собой заголовок раздела. Элемент используется для группировки набора элементов h1—h6, в тех случаях, когда заголовок имеет множество уровней, включая подзаголовки, альтернативные заголовки, или строки тегов (taglines)

Таблица 3.1 *(окончание)*

Элемент	Описание
`<header>`	Элемент `header` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-header-element**) представляет группу вспомогательных инструментов — вводных или навигационных. Элемент `header` обычно предназначается для того, чтобы содержать заголовок раздела (элемент `h1`—`h6` или элемент `hgroup`), но это требование не является обязательным. Элемент `header` также может использоваться для "обертывания" оглавления раздела, формы поиска или любых логотипов
`<footer>`	Элемент `footer` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-footer-element**) представляет собой заголовок для ближайшего дочернего раздела или для корневого элемента раздела. Нижний колонтитул (`footer`) обычно содержит информацию о своем разделе, в частности, сведения об авторе текста, ссылки на другие документы, связанные с данным, информацию об авторских правах и так далее. Нижние колонтитулы не обязательно должны размещаться в конце раздела, хотя обычно это бывает именно так. Когда элемент нижнего колонтитула содержит целые разделы, эти разделы обычно представляют собой приложения к документу, длинных выходных данных издательства (colophons), многословные лицензионные соглашения и прочую аналогичную информацию
`<time>`	Элемент `time` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/text-level-semantics.html#the-time-element**) представляет либо время по 24-часовой шкале, либо точную дату по григорианскому календарю (по новому стилю), обычно с указанием времени и часового пояса с соответствующим сдвигом
`<mark>`	Элемент `mark` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/text-level-semantics.html#the-mark-element**) представляет фрагмент текста в документе, помеченный цветом или выделенный подсветкой для справочных целей

Я понимаю, что вы с нетерпением ожидаете того момента, когда вы сможете начать пользоваться этими новыми элементами, иначе зачем бы вы стали читать данную главу. Но в первую очередь нам необходимо сделать небольшое отступление.

Отступление, описывающее, как браузеры обрабатывают неизвестные элементы

Каждый браузер имеет мастер-список элементов HTML, которые он поддерживает. Например, список поддерживаемых элементов Mozilla Firefox хранится в файле `nsElementTable.cpp` (см. **https://mxr.mozilla.org/seamonkey/source/parser/htmlparser/src/nsElementTable.cpp**). Элементы, не перечисленные в этом списке, обрабатываются как "неизвестные элементы" (unknown elements"). С неизвестными элементами связаны две фундаментальные проблемы:

1. *Каким должен быть стиль элемента?* По умолчанию, `<p>` имеет отступы сверху и снизу, `<blockquote>` имеет отступ от левой границы, а `<h1>` отображается

более крупным шрифтом. Но какие стили следует применять к неизвестным элементам?

2. *Как должен выглядеть DOM-объект элемента?* Файл Mozilla `nsElementTable.cpp` содержит информацию о том, какие типы других элементов может содержать данный элемент. Если вы включаете разметку наподобие `<p><p>`, то элемент второго абзаца неявно закрывает первый, так что элементы выглядят как параллельные "отростки" (siblings), а не состоят в отношениях "родитель-потомок". Но если вы напишете `<p><span>`, то элемент `span` не закрывает абзац, потому что браузеру Firefox известно о том, что `<p>` — это блочный элемент, который может содержать встроенный (inline) элемент `<span>`. Таким образом, элемент `<span>` в DOM оказывается потомком элемента `<p>`.

Различные браузеры по-разному решают эти вопросы. Да, я знаю, что это может шокировать. Из всех популярных браузеров, Microsoft Internet Explorer дает на этот вопрос наиболее проблематичный ответ, но в данной области каждому браузеру требуется небольшая помощь.

Ответить на первый вопрос относительно просто: не нужно присваивать никакого особенного стиля форматирования неизвестным элементам. Пусть они наследуют любые действующие свойства CSS везде, где они встречаются на странице, и пусть автор страницы имеет полную свободу указания стилей с помощью CSS. И в большинстве случаев этот подход работает, но существует одна тонкость, о которой вам необходимо знать.

Замечание Профессора Разметки

Все браузеры визуализируют неизвестные элементы как встроенные (inline), т. е. так, как если бы они имели правило CSS `display:inline`.

Существует несколько новых элементов, определенных в HTML5, являющихся элементами блочного уровня. Это значит, что они могут, в свою очередь, содержать другие элементы блочного уровня, и браузеры, соответствующие требованиям HTML5, по умолчанию будут считать, что они имеют стиль `display:block`. Если вы хотите использовать эти элементы в более старых браузерах, вам потребуется определять стиль отображения вручную, как показано в листинге 3.4.

Листинг 3.4. Ручное определение стиля элементов для старых браузеров

```
article,aside,details,figcaption,figure,
footer,header,hgroup,menu,nav,section {
    display:block;
}
```

Примечание

Код из листинга 3.4 позаимствован из таблицы стилей HTML5 Reset Stylesheet (см. **http://html5doctor.com/html-5-reset-stylesheet/**), написанной Ричардом Кларком (Rich Clark). Упомянутая таблица стилей делает и еще многое другое, помимо задач, описанных в данной главе.

Но постойте, ситуация ухудшается! До выхода версии 9, в Internet Explorer неизвестные элементы вообще *не получали никаких стилей*. Например, представьте себе, что у вас была разметка, примерно приведенная в листинге 3.5.

Листинг 3.5. Пример разметки с неизвестными элементами

```
<style type="text/css">
  article { display: block; border: 1px solid red }
</style>
...
<article>
<h1>Welcome to Initech</h1>
<p>This is your <span>first day</span>.</p>
</article>
```

Internet Explorer (вплоть до версии IE 8 включительно) не станет трактовать элемент `<article>` как элемент блочного уровня, не будет и отображать красную рамку вокруг статьи. Все правила стилей попросту игнорируются. На тот момент, когда я пишу эти строки, Internet Explorer 9 все еще находится на стадии бета-тестирования (**http://blogs.msdn.com/b/ie/archive/2009/11/18/an-early-look-at-ie9-for-developers.aspx**), но корпорация Microsoft уже объявила (а разработчики подтвердили), что в Internet Explorer 9 этой проблемы уже не будет[1].

Вторая проблема — это DOM-объект, который создают браузеры, когда им встречаются неизвестные элементы. Здесь, как всегда, наибольшее количество проблем создает Internet Explorer. Если IE явно не распознает имя элемента, он вставит в DOM-объект элемент *в виде пустого узла, не имеющего потомков*. Все элементы, от которых вы ожидаете, что они будут прямыми потомками неизвестного элемента, фактически будут вставлены как побочные ветви (siblings).

В листингах 3.6 и 3.7 представлена диаграмма в символах ASCII, которая иллюстрирует это отличие. DOM-объект в том виде, в котором его подразумевает "видеть" стандарт HTML5, показан в листинге 3.6.

Листинг 3.6. DOM-объект в том виде, в котором его подразумевает "видеть" стандарт HTML5

```
article
|
+--h1 (child of article)
|  |
|  +--text node "Welcome to Initech"
|
```

[1] Тем не менее, в версии Internet Explorer 9 Release Candidate (сборка 9.0.8080.16413) проблема пока еще остается. — *Прим. перев.*

```
+--p (child of article, sibling of h1)
   |
   +--text node "This is your "
   |
   +--span
   |  |
   |  +--text node "first day"
   |
   +--text node "."
```

В листинге 3.7 представлен DOM-объект, который на практике будет создан браузером Internet Explorer.

Листинг 3.7. DOM-объект, который на практике создает Internet Explorer:

```
article (no children)
h1 (sibling of article)
|
+--text node "Welcome to Initech"
p (sibling of h1)
|
+--text node "This is your "
|
+--span
|  |
|  +--text node "first day"
|
+--text node "."
```

Для этой проблемы существует замечательный "обходной" путь. Если вы создадите "болванку" элемента `<article>` с помощью JavaScript перед тем, как фактически начнете использовать вашу страницу (см. **http://xopus.com/devblog/2008/style-unknown-elements.html**), то Internet Explorer магическим образом распознает ваш элемент `<article>` и позволит вам наложить на него стили при помощи CSS. Поэтому нет необходимости вставлять "болванку" элемента в DOM. Просто создайте элемент один раз (на страницу), и этого будет достаточно, чтобы "научить" Internet Explorer накладывать стили на элементы, которые этот браузер не распознает, как показано в листинге 3.8.

Листинг 3.8. "Обучение" Internet Explorer наложению стилей на нераспознаваемые элементы

```
<html>
<head>
<style>
```

```
  article { display: block; border: 1px solid red }
</style>
<script>document.createElement("article");</script>
</head>
<body>
<article>
<h1>Welcome to Initech</h1>
<p>This is your <span>first day</span>.</p>
</article>
</body>
</html>
```

Этот код будет работать во всех версиях Internet Explorer, вплоть до IE 6! Мы можем расширить этот прием с тем, чтобы создавать "копии-болванки" для всех новых элементов HTML5 одновременно — опять же, они никогда не вставляются в DOM, так что вы никогда не увидите этих фиктивных элементов — а затем просто начать пользоваться ими, не слишком беспокоясь о браузерах, которые не обеспечивают поддержки HTML5.

Реми Шарп (Remy Sharp) проделал именно это в коде своего скрипта, активизирующего поддержку HTML5 (**http://remysharp.com/2009/01/07/html5-enabling-script/**). На момент написания этой книги скрипт выдержал уже 14 пересмотров, но его основная идея иллюстрируется листингом 3.9.

Листинг 3.9. Ключевой фрагмент скрипта Реми Шарпа, активизирующего поддержку HTML5

```
<!--[if lt IE 9]>
<script>
  var e = ("abbr,article,aside,audio,canvas,datalist,details," +
    "figure,footer,header,hgroup,mark,menu,meter,nav,output," +
    "progress,section,time,video").split(',');
  for (var i = 0; i < e.length; i++) {
    document.createElement(e[i]);
  }
</script>
<![endif]-->
```

Элементы `<!--[if lt IE 9]>` и `<![endif]-->` представляют собой условные комментарии (см. **http://msdn.microsoft.com/en-us/library/ms537512%28VS.85%29.aspx**). Но Internet Explorer интерпретирует их как утверждение: "если текущий браузер представляет собой версию Internet Explorer, более раннюю, чем v.9, тогда следует исполнять этот блок". Любой другой браузер будет трактовать весь блок как комментарий HTML. "Сухой остаток" заключается в том, что Internet Explorer (вплоть

до версии 8 включительно) будет исполнять данный скрипт, а все остальные браузеры будут его полностью игнорировать. Благодаря этому ваша страница будет быстрее загружаться в тех браузерах, которым не нужен данный "хак".

Сам по себе код на JavaScript относительно прост. Переменная `e` превращается в массив строк наподобие `abbr`, `article`, `aside`, и т. д. Затем исполняется цикл, в котором из этих строк создаются именованные элементы при помощи вызовов функции `document.createElement()`. Но, поскольку мы игнорируем возвращаемое значение, элементы никогда не вставляются в DOM. Но этого достаточно, чтобы заставить Internet Explorer обрабатывать эти элементы так, как требуется нам, поскольку впоследствии мы будем их использовать в тексте страницы.

Это "впоследствии" на самом деле очень важно. Данный скрипт должен быть помещен в самое начало вашей страницы, предпочтительно, в элемент `<head>`, а не в ее конец. Благодаря такому размещению, Internet Explorer исполнит скрипт *прежде*, чем начнет разбор ваших тегов и атрибутов. Если данный скрипт поместить в конец страницы, то Internet Explorer доберется до него слишком поздно. К тому моменту, когда это произойдет, Internet Explorer уже успеет интерпретировать вашу разметку, сделает это неправильно, построит неправильный DOM-объект, и, несмотря на исполнение скрипта, браузер не возвратится назад, чтобы исправить ошибку.

Реми Шарп (Remy Sharp) "минимизировал" этот скрипт и разместил его на хостинге Google (**https://code.google.com/p/html5shiv/**). На тот случай, если вы заинтересовались: сам скрипт представляет собой открытое ПО (open source) и лицензирован MIT, так что вы вправе использовать его в любом из своих проектов. Если вы предпочитаете, вы можете даже дать на этот скрипт "горячую ссылку", указав на версию, размещенную на хостинге, например, так, как показано в листинге 3.10.

Листинг 3.10. Пример использования скрипта Реми Шарпа в коде вашей страницы

```
<head>
  <meta charset="utf-8" />
  <title>My Weblog</title>

  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>

</head>
```

Теперь мы готовы начать пользование новыми семантическими элементами HTML5.

Заголовки

Вернемся обратно к странице, на примере которой мы ведем рассмотрение материала данной главы. В частности, давайте рассмотрим как раз заголовки, представленные в листинге 3.11.

Листинг 3.11. Фрагменты исходного кода страницы-примера

```
<div id="header">
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
</div>

…

<div class="entry">
  <h2>Travel day</h2>
</div>

…

<div class="entry">
  <h2>I'm going to Prague!</h2>
</div>
```

В этой разметке нет ничего особенного. Если она вам нравится, можете ее сохранить. Она вполне приемлема и для HTML5. Но HTML5 предоставляет некоторые дополнительные семантические элементы для заголовков и разделов.

Во-первых, теперь можно избавиться от `<div id="header">`. Это — типичный часто встречающийся фрагмент, но при этом данный фрагмент ничего не значит! Элемент `div` не имеет определенного смысла, а атрибут `id` не имеет никакого семантического определения (пользовательским агентам не разрешается привносить никакого значения из значений атрибута `id`). Вы могли бы заменить это значение следующим: `<div id="shazbot">`, и оно имело бы тот же самый семантический смысл, иначе говоря, вообще никакого.

HTML5 для этой цели определяет элемент `<header>`. Спецификация HTML5 содержит практические примеры использования элемента `<header>` (см. **http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-header-element**). Вот как он выглядел бы на нашей странице с примером (листинг 3.12).

Листинг 3.12. Пример использования элемента `<header>`

```
<header>
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
  …
</header>
```

Это хорошо. Это сообщает всем заинтересованным лицам о том, что данный элемент представляет собой заголовок. Но как обстоят дела со строкой тегов? Это — еще один общий "шаблон", который вплоть до настоящего времени не имел стандартной разметки. Построение строки тегов — довольно сложная вещь для создания разметки. Строка тегов (tagline) напоминает подзаголовок, но она "присоединена" к основному заголовку. Это значит, что она представляет собой подзаголовок, не создающий собственного раздела.

Элементы заголовков наподобие `<h1>` и `<h2>` придают вашей странице определенную структуру. Если их рассматривать совместно, то они создают "контур", который можно использовать для визуализации страницы (или для навигации по ней). Экранные заголовки используют разметку документа с тем, чтобы помогать слабовидящим пользователям перемещаться по странице. Существуют специальные онлайновые инструменты (**http://gsnedders.html5.org/outliner/**) и браузерные расширения (**http://chrispederick.com/work/web-developer/**), которые могут помочь вам визуализировать ваши документы в режиме онлайн.

В HTML 4, элементы `<h1>`–`<h6>` представляют собой *единственный* способ создания документа в оперативном режиме. Структура документа на нашей странице с примером выглядит так, как показано в листинге 3.13.

Листинг 3.13. Структура примера в HTML 4

```
My Weblog (h1)
   |
   +--Travel day (h2)
   |
   +--I'm going to Prague! (h2)
```

Это замечательно, но это же означает, что у нас нет способа разметить строку тегов "A lot of effort went into making this effortless" ("На то, чтобы эта задача стала простой, пришлось затратить очень много труда"). Если мы попытаемся разметить ее с помощью `<h2>`, то мы добавим в структуру документа "фантомный узел", как видно в листинге 3.14.

Листинг 3.14. Появление "фантомного узла" в структуре документа

```
My Weblog (h1)
   |
   +--A lot of effort went into making this effortless. (h2)
   |
   +--Travel day (h2)
   |
   +--I'm going to Prague! (h2)
```

Но это не отражает структуры документа. Строка тегов (tagline) не представляет отдельного раздела; она является просто подзаголовком.

Возможно, мы могли бы разметить строку тегов с помощью <h2> и пометить заголовок каждой статьи с помощью <h3>? Нет, это будет еще хуже, что и видно из листинга 3.15.

Листинг 3.15. Попытка пометить заголовок каждой статьи с помощью <h3> ухудшает ситуацию

```
My Weblog (h1)
   |
   +--A lot of effort went into making this effortless. (h2)
         |
         +--Travel day (h3)
         |
         +--I'm going to Prague! (h3)
```

Теперь у нас по-прежнему есть "фантомный узел" в структуре документа, но он дополнительно еще и "украл" потомков, которые по праву принадлежат корневому узлу. Отсюда вытекает проблема: HTML 4 не предоставляет способа разметки подзаголовка без его добавления в структуру документа. Не имеет значения, каким образом мы попытаемся обойти эту проблему, но подзаголовок "A lot of effort went into making this effortless" в любом случае окажется в низу этого графа. И именно поэтому мы закончили семантически бессмысленной разметкой наподобие <p class="tagline">.

HTML5 предоставляет решение для этой проблемы: элемент <hgroup>. Элемент <hgroup> работает как "обертка" для двух или большего количества *связанных* заголовочных элементов. Каково в данном случае значение слова "связанных"? На самом деле это значит, что они образуют единый узел в структуре документа.

Например, рассмотрим разметку, представленную в листинге 3.16.

Листинг 3.16. Пример разметки строки тегов с помощью элемента <hgroup>

```
<header>
  <hgroup>
    <h1>My Weblog</h1>
    <h2>A lot of effort went into making this effortless.</h2>
  </hgroup>
  …
</header>

…

<div class="entry">
  <h2>Travel day</h2>
```

```
</div>

…

<div class="entry">
  <h2>I'm going to Prague!</h2>
</div>
```

А вот структура документа, созданная с помощью этой разметки (листинг 3.17).

Листинг 3.17. Структура документа, созданного с помощью элемента <hgroup>

```
My Weblog (h1 of its hgroup)
   |
   +--Travel day (h2)
   |
   +--I'm going to Prague! (h2)
```

Вы можете протестировать собственные страницы, воспользовавшись инструментом HTML5 Outliner (**http://gsnedders.html5.org/outliner/**), чтобы убедиться в том, что вы правильно используете заголовочные элементы.

Статьи (Articles)

Продолжим изучение нашего примера, но теперь давайте рассмотрим вариант разметки, представленный в листинге 3.18.

Листинг 3.18. Пример, иллюстрирующий разметку статьи на странице

```
<div class="entry">
  <p class="post-date">October 22, 2009</p>
  <h2>
    <a href="#"
       rel="bookmark"
       title="link to this post">
       Travel day
    </a>
  </h2>
  …
</div>
```

Как и в предыдущем случае, это — легальный код HTML5. Но HTML5 предоставляет и специализированный элемент для более общего случая разметки статьи

на странице — этот элемент вполне предсказуемо получил название `<article>`. Использование элемента `<article>` продемонстрировано в листинге 3.19.

Листинг 3.19. Пример, иллюстрирующий разметку статьи на странице с помощью элемента <article>

```
<article>
  <p class="post-date">October 22, 2009</p>
  <h2>
    <a href="#"
       rel="bookmark"
       title="link to this post">
       Travel day
    </a>
  </h2>
  …
</article>
```

Да, но здесь не все так уж просто. Есть еще одно изменение, которое нам необходимо внести. Сначала я его продемонстрирую (листинг 3.20), а потом поясню.

Листинг 3.20. Еще одно изменение, которое необходимо внести в разметку статьи при использовании элемента <article>

```
<article>
  <header>
    <p class="post-date">October 22, 2009</p>
    <h1>
      <a href="#"
         rel="bookmark"
         title="link to this post">
         Travel day
      </a>
    </h1>
  </header>
  …
</article>
```

Вы улавливаете смысл? Я заменил элемент `<h2>` элементом `<h1>` и "обернул" всю эту конструкцию элементом `<header>`. Вы уже видели элемент `<header>` в действии. Его цель заключается в том, чтобы служить "оберткой" для всех элементов, которые служат заголовком статьи (в данном случае, это название статьи и дата ее публикации). Но…но…но… разве не полагается иметь только один элемент `<h1>`

на документ? Не покорежит ли это структуру документа? Нет, но для того, чтобы понять, почему этого не произойдет, нам нужно вернуться на шаг назад.

В HTML 4, единственным способом создания структуры документа служили элементы <h1>—<h6>. Если вам хотелось иметь в структуре вашего документа только один корневой узел, вам следовало ограничиться только одним элементом <h1> на всю разметку вашего документа. Но в спецификации HTML5 определяется алгоритм (**http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#headings-and-sections**) для генерации структуры документа, который включает в свой состав новые семантические элементы HTML5. Алгоритм HTML5 говорит о том, что элемент <article> создает новый раздел, а именно новый узел в структуре документа. И в HTML5 каждый раздел может иметь собственный элемент <h1>.

Это — радикальное изменение по сравнению с HTML 4, и это изменение весьма позитивно. Многие Web-страницы в действительности генерируются по шаблонам. Часть содержимого берется из одного источника и вставляется в одном местоположении в пределах страницы; часть содержимого берется из другого источника и вставляется в другом местоположении в пределах той же страницы. Многие инструкции структурируются таким же способом: "Вот некоторая информация, размеченная по правилам HTML. Просто скопируйте ее и вставьте в вашу страницу." Это неплохо для небольших элементов электронного контента, но что, если разметка, которую вы вставляете, представляет собой целый раздел? В этом случае инструкция может выглядеть примерно так: "Вот фрагмент информации, размеченный по правилам HTML. Скопируйте ее, вставьте в любой текстовый редактор, исправьте теги заголовков так, чтобы они соответствовали уровню вложения соответствующих заголовков на странице, на которую вы собираетесь его вставить".

Давайте рассмотрим изложенное с другой позиции. HTML 4 не имеет родового (*generic*) элемента заголовка. Там есть только шесть пронумерованных элементов заголовка, <h1>—<h6>, которые должны быть вложены друг в друга в точном соответствии с этим порядком. И это плохо, особенно если ваша страница "собирается" из готовых кусков, а не "составляется", как авторская страница. И это та проблема, которую HTML5 решает с помощью новых элементов разбиения на разделы и новых правил для существующих заголовочных элементов. Если вы применяете новые заголовочные элементы, я могу предложить вам разметку, представленную в листинге 3.21.

Листинг 3.21. Вариант разметки с использованием новых семантических элементов HTML5

```
<article>
  <header>
    <h1>A syndicated post</h1>
  </header>
  <p>Lorem ipsum blah blah…</p>
</article>
```

Вы можете скопировать эту разметку и вставить ее куда угодно в пределах вашей страницы без модификации. Тот факт, что она содержит элемент <h1>,

не представляет проблемы, потому что вся разметка обрамляется элементами <article>. Элемент <article> определяет самодостаточный узел в структуре документа, элемент <h1> предоставляет для этого узла заголовок, а все остальные элементы, служащие для разбиения на разделы, останутся на тех уровнях, на которых они находились раньше.

Замечания Профессора Разметки

Как и в случае с другими вещами в Web, реальность всегда несколько сложнее, чем ее описывают. Новые элементы "явного" разбиения на разделы (как элемент <h1>, обрамленный элементами <article>) могут неожиданным образом взаимодействовать со старыми, "неявными" элементами разбиения на разделы (<h1>–<h6> как таковыми). Ваша жизнь станет намного проще, если вы будете использовать только один из них, но не оба. Если вы должны использовать оба типа элементов на одной странице, проверьте полученный результат в HTML5 Outliner и убедитесь в том, что структура вашего документа вполне осмысленна.

Даты и времена

Это вдохновляет, не правда ли? Давайте вернемся к нашей странице с примером. Следующая строка, которую я хочу выделить, выглядит так, как показано в листинге 3.22.

Листинг 3.22. Семантическая разметка даты публикации

```
<div class="entry">
  <p class="post-date">October 22, 2009</p>
  <h2>Travel day</h2>
</div>
```

Та же самая старая история, так? Общий шаблон — назначение даты публикации статьи, которая не имеет семантической разметки для резервного копирования, так что авторам приходится прибегать к типовой разметке без индивидуальных атрибутов класса. Здесь, как и ранее, мы имеем корректный код HTML5. Вам нет никакой *необходимости* его менять. Но HTML5 предоставляет для этого случая специальное решение: элемент <time>.

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

Существуют три части элемента <time>:

- временная отметка (timestamp) в формате, удобном для машинной интерпретации;
- текстовое содержимое в формате, удобном для восприятия человеком;
- необязательный флаг pubdate.

В рассматриваемом примере атрибут datetime указывает только дату, без указания времени. Формат подразумевает указание четырех цифр для года, двух цифр

для месяца, и двух цифр — для указания дня, причем в качестве разделителя используется дефис:

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

Если вы хотите указывать и время, добавьте букву `T` после даты, затем укажите время в 24-часовом формате, а затем — часовой пояс, как показано в листинге 3.23.

Листинг 3.23. Пример использования атрибута `datetime` с указанием даты и времени

```
<time datetime="2009-10-22T13:59:47-04:00" pubdate>
  October 22, 2009 1:59pm EDT
</time>
```

Формат даты и времени довольно гибок. В спецификации HTML5 приведены примеры корректных строк, указывающих дату и время (**http://www.whatwg.org/specs/web-apps/current-work/multipage/common-microsyntaxes.html#valid-global-date-and-time-string**).

Обратите внимание на то, что я изменил текстовое содержимое — информацию, заключенную между тегами `<time>` и `</time>` таким образом, чтобы оно соответствовало формату временной отметки (timestamp), как ее воспринимает машина. На самом деле, необходимости в этом нет. Текстовое содержимое может быть таким, как хочется вам, до тех пор, пока вы указываете атрибут `datetime` в формате, подходящем для машинной интерпретации. Поэтому следующий код вполне корректен с позиций HTML5:

```
<time datetime="2009-10-22">last Thursday</time>
```

И этот код тоже представляет собой корректный код HTML5:

```
<time datetime="2009-10-22"></time>
```

Заключительный фрагмент головоломки — это атрибут `pubdate`. Это — атрибут типа `Boolean`, так что при необходимости его можно просто добавить следующим образом:

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

Если вам не нравятся "голые" атрибуты, то можно пользоваться следующим эквивалентом:

```
<time datetime="2009-10-22" pubdate="pubdate">October 22, 2009</time>
```

Что означает атрибут `pubdate`? Две вещи. Если элемент `<time>` содержится внутри элемента `<article>`, то это означает, что данная временная отметка (timestamp) представляет собой дату публикации статьи. Если элемент `<time>` не находится внутри элемента `<article>`, то это значит, что временная отметка относится к дате публикации всего документа.

Рассмотрим пример целой статьи, переформулированный таким образом, чтобы в полной мере использовать преимущества HTML5. Этот пример представлен в листинге 3.24.

Листинг 3.24. Пример целой статьи, переформулированный таким образом, чтобы в полной мере использовать преимущества HTML5

```
<article>
  <header>
    <time datetime="2009-10-22" pubdate>
      October 22, 2009
    </time>
    <h1>
      <a href="#"
         rel="bookmark"
         title="link to this post">
         Travel day
      </a>
    </h1>
  </header>
  <p>Lorem ipsum dolor sit amet…</p>
</article>
?
```

Навигация

Одним из наиболее важных элементов любого Web-сайта является панель навигации. Сайт CNN.com использует "вкладки", расположенные вдоль верхней границы окна браузера, которые отсылают посетителя к разным разделам новостей, таким, как "Технологии", "Здоровье", "Спорт" и т. д. Страницы поиска Google имеют аналогичную "ленту" в верхней части страницы, чтобы вы могли попробовать повторить свой поиск по другим категориям информационного наполнения — таким, как "Картинки" (Images), "Видео" (Video), "Карты" (Maps) и т. д. И наша страница с примером тоже имеет навигационную панель в виде заголовка, в которой можно найти ссылки, отсылающие к различным разделам нашего гипотетического сайта — "home", "blog", "gallery" и "about".

В листинге 3.25 показано, как наша навигационная панель была размечена сначала.

Листинг 3.25. Изначальный вариант разметки навигационной панели тестового сайта

```
<div id="nav">
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</div>
```

Опять же, этот код вполне корректен и для HTML5. Но, хотя он размечен, как список из четырех ссылок, в нем нет ничего, что указывало бы вам на то, что он является элементом системы навигации. При визуальном рассмотрении, можно предположить, что это так, потому что он является частью заголовка страницы, а также по прочтении текста ссылок. Но семантически ничто не дает нам оснований отличить эти ссылки от любых других.

Кому вообще надо заботиться о семантическом смысле элементов навигации по сайту? Во-первых — тем, кто заботится о посетителях с ограниченными физическими возможностями (**http://diveintoaccessibility.org/**). Почему? Рассмотрим следующий сценарий: предположим, пользователь ограничен в движениях, и пользоваться мышью для него затруднительно, если вообще возможно. В качестве компенсации можно предложить дополнительный модуль браузера (add-on), который позволил бы вам переходить вперед или назад по большинству ссылок навигации. Или давайте рассмотрим следующий сценарий: вы плохо видите, вам было бы лучше использовать специализированную программу типа "экранной читалки" (screenreader), которая зачитывала бы вслух содержимое Web-страниц и преобразовала бы его в краткую сводку, используя для этого преобразование текста в речь с применением "синтеза голоса". Как только вы минуете заголовок страницы, следующей важной частью содержащейся на ней информации будут навигационные ссылки (navigation links). Если вам требуется быстрая навигация, вы сообщаете программе чтения с экрана (screenreader) о необходимости перехода к панели навигации и начале зачитывания. Если вы хотите перейти к быстрому просмотру, вы можете дать программе screenreader команду *пропустить* панель навигации и начать зачитывание основного содержимого. В любом случае, возможность программного определения навигационных ссылок крайне важна.

Таким образом, будет правильно и вполне корректно, если вы определите разметку навигации по вашему сайту с помощью элементов `<div id="nav">`, но, наряду с этим, здесь нет ничего и особо прогрессивного. С точки зрения обычных людей этот подход далек от оптимального. HTML5 предлагает семантический подход к разметке навигационных разделов: элемент `<nav>`. Его использование продемонстрировано в листинге 3.26.

Листинг 3.26. Использование элемента <nav>

```
<nav>
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</nav>
```

Вопрос к Профессору Разметки

В: Совместимы ли skip-ссылки (**http://webaim.org/techniques/skipnav/**) с элементом <nav>? Нужно ли мне читать эти ссылки в HTML5?

О: Ссылки пропуска (skip-ссылки) позволяют читателям пропустить разделы навигации. Они полезны для пользователей с ограниченными физическими возможностями, использующих стороннее ПО для зачитывания вслух содержимого Web-страницы и для навигации по ней без использования мыши.

Когда ПО для чтения содержимого с экрана обновлено так, чтобы получить возможность распознавания элементов <nav>, skip-ссылки станут ненужными, потому что ПО для чтения вслух экранного содержимого "научится" автоматически "перешагивать" через навигационные разделы, размеченные элементами <nav>. Тем не менее, должно пройти некоторое время, прежде чем все пользователи, нуждающиеся в ПО для зачитывания содержимого экрана, смогут обновить его до версий, поддерживающих HTML5, поэтому вам необходимо пока продолжать определять собственные ссылки пропуска (skip links), позволяющие пропускать разделы <nav>.

Нижние колонтитулы (Footers)

Наконец, мы добрались до конца нашей страницы с примером. Последней из тем, которые мы с вами обсудим в этой главе, будет заключительная часть страницы, которую вы видите — нижний колонтитул. Изначально нижний колонтитул был размечен так, как показано в листинге 3.27.

Листинг 3.27. Изначальный вариант разметки нижнего колонтитула тестового примера

```
<div id="footer">
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</div>
```

Это — легальный код HTML5. Если он вас устраивает, вы можете ничего не менять. Но HTML5 предоставляет для реализации той же задачи более специализированный элемент: <footer>. Его использование демонстрируется в листинге 3.28.

Листинг 3.28. Разметка нижнего колонтитула с использованием элемента <footer>

```
<footer>
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</footer>
```

Какую информацию целесообразно размечать с применением элемента `<footer>`? Вероятно, все то, что до сих пор выделялось при помощи `<div id="footer">`. OK, это "закольцованный ответ". Но это и в самом деле то, что я хотел вам сказать. Дела обстоят именно так. В спецификации HTML5 говорится:

> "Нижний колонтитул (footer) обычно содержит информацию о разделе, к которому он принадлежит, включая сведения об авторе, написавшем материал, ссылки на документы, имеющие отношение к данному материалу, информацию об авторских правах и т. д.".

Все это как раз и содержится в колонтитуле нашей страницы с примером: краткое уведомление об авторских правах и ссылка на страницу, содержащую информацию об авторе. Просматривая ряд популярных сайтов с высокой посещаемостью, я обнаружил, что элемент `footer` имеет очень высокий потенциал.

- **CNN** имеет нижний колонтитул, который содержит информацию об авторских правах, ссылки на переводы, а также ссылки на страницы, где приводятся условия пользования сервисом, соглашения о конфиденциальности, а также страницы наподобие "about us", "contact us" и "help". Все это, безусловно, представляет собой материал, предназначенный для приведения именно в нижнем колонтитуле, определяемом элементом `<footer>`.
- **Google** имеет домашнюю страницу, знаменитую своим минимализмом, но даже там есть нижняя строка со ссылками типа "Advertising Programs", "Business Solutions" и "About Google"; плюс уведомление об авторских правах; плюс ссылка на страницу, где изложены условия политики конфиденциальности, которых придерживается Google. Все это вполне заслуживает того, чтобы быть размеченным с помощью элемента `<footer>`.
- **My weblog**, наша страница с примером, тоже имеет нижний колонтитул, содержащий ссылки на другие сайты, плюс уведомление об авторских правах. Определенно, все это можно размечать с помощью элемента `<footer>`. Обратите внимание на то, что сами ссылки *не должны* быть заключены в элементы `<nav>`, потому что они не являются ссылками навигации по сайту; они просто представляют собой коллекцию ссылок на мои материалы, размещенные на других сайтах.

"Толстые нижние колонтитулы" в настоящее время широко распространены. Например, взгляните на нижний колонтитул сайта W3C. Он состоит из трех столбцов, с метками "Navigation", "Contact W3C" и "W3C Updates". Разметка выглядит примерно так, как показано в листинге 3.29.

Листинг 3.29. Фрагмент разметки нижнего колонтитула сайта W3C

```
<div id="w3c_footer">
  <div class="w3c_footer-nav">
    <h3>Navigation</h3>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/standards/">Standards</a></li>
      <li><a href="/participate/">Participate</a></li>
      <li><a href="/Consortium/membership">Membership</a></li>
      <li><a href="/Consortium/">About W3C</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>Contact W3C</h3>
    <ul>
      <li><a href="/Consortium/contact">Contact</a></li>
      <li><a href="/Help/">Help and FAQ</a></li>
      <li><a href="/Consortium/sup">Donate</a></li>
      <li><a href="/Consortium/siteindex">Site Map</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>W3C Updates</h3>
    <ul>
      <li><a href="http://twitter.com/W3C">Twitter</a></li>
      <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
    </ul>
  </div>
  <p class="copyright">Copyright © 2009 W3C</p>
</div>
```

Чтобы преобразовать эту разметку в семантическую разметку HTML5, я бы внес следующие изменения:

- преобразование внешнего элемента `<div id="w3c_footer">` в элемент `<footer>`;
- преобразование первых двух экземпляров `<div class="w3c_footer-nav">` в элементы `<nav>`, а третьего экземпляра — в элемент `<section>`;
- преобразование заголовков `<h3>` в заголовки `<h1>`, поскольку теперь каждый из них будет находиться внутри элемента, выполняющего разбиение на разделы. Элемент `<nav>` создает раздел в структуре документа, так же, как и элемент `<article>`.

Окончательный вариант разметки может выглядеть так, как показано в листинге 3.30.

Листинг 3.30. Предложенный автором вариант разметки нижнего колонтитула сайта W3C

```
<footer>
  <nav>
    <h1>Navigation</h1>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/standards/">Standards</a></li>
      <li><a href="/participate/">Participate</a></li>
      <li><a href="/Consortium/membership">Membership</a></li>
      <li><a href="/Consortium/">About W3C</a></li>
    </ul>
  </nav>
  <nav>
    <h1>Contact W3C</h1>
    <ul>
      <li><a href="/Consortium/contact">Contact</a></li>
      <li><a href="/Help/">Help and FAQ</a></li>
      <li><a href="/Consortium/sup">Donate</a></li>
      <li><a href="/Consortium/siteindex">Site Map</a></li>
    </ul>
  </nav>
  <section>
    <h1>W3C Updates</h1>
    <ul>
      <li><a href="http://twitter.com/W3C">Twitter</a></li>
      <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
    </ul>
  </section>
  <p class="copyright">Copyright © 2009 W3C</p>
</footer>
```

Материалы для дополнительного чтения

Примеры страниц, использованных в этой главе:

- Оригинал (HTML 4) — **http://diveintohtml5.org/examples/blog-original.html**
- Модифицированная страница (HTML5) — **http://diveintohtml5.org/examples/blog-html5.html**

О кодировке символов:

- *The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)*, Статья Джоэла Сполски (Joel Spolsky) — **http://www.joelonsoftware.com/articles/Unicode.html**
- *On the Goodness of Unicode* (**http://www.tbray.org/ongoing/When/200x/2003/04/06/Unicode**), *On Character Strings* (**http://www.tbray.org/ongoing/When/200x/2003/04/13/Strings**), и *Characters vs. Bytes* (**http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF**) — статьи Тима Брея (Tim Bray)

Об использовании новых функций HTML5 в Internet Explorer:

- *How to style unknown elements in IE* (**http://xopus.com/devblog/2008/style-unknown-elements.html**) — статья Сьорда Висшера (Sjoerd Visscher)
- *HTML5 shiv* (**http://ejohn.org/blog/html5-shiv/**) — статья Джона Резига (John Resig)
- *HTML5 enabling script* (**http://remysharp.com/2009/01/07/html5-enabling-script/**) — статья Реми Шарпа (Remy Sharp)

О режимах поддержки стандартов и перехвате типа документа (doctype sniffing):

- *Activating Browser Modes with Doctype* (**http://hsivonen.iki.fi/doctype/**) — статья Генри Сайвонена (Henri Sivonen). Это единственная статья, которую вам обязательно нужно прочесть по данному вопросу. Все остальные статьи, посвященные типам документа (doctypes) и не ссылающиеся на работу Генри, гарантированно устарели, либо неполны, либо просто ошибочны.

О проверке правильности кода с поддержкой HTML5 (HTML5-aware validator):

- **html5.validator.nu**

Глава 4

Давайте назовем это "холстом" (поверхностью для рисования)

HTML5 определяет элемент `<canvas>` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html**) как "зависящую от экранного разрешения область для размещения битовой карты (bitmap), которая может использоваться для визуализации диаграмм, графиков, игровой графики или других изображений "на лету" (on the fly)". *Холст* (canvas) представляет собой прямоугольную область на вашей странице, в пределах которой вы можете создавать любые изображения с применением JavaScript.

В табл. 4.1 описана базовая поддержка элемента `<canvas>` наиболее распространенными браузерами и мобильными устройствами.

Таблица 4.1. *Базовая поддержка элемента* `<canvas>` *популярными браузерами и мобильными устройствами*

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Для поддержки <canvas> в Internet Explorer требуется библиотека стороннего разработчика explorercanvas (**https://code.google.com/p/explorercanvas/**).

И как же выглядит этот "холст"? В действительности — вообще никак. Элемент `<canvas>` не имеет ни собственного содержимого, ни собственной рамки.

Его разметка выглядит так:

```
<canvas width="300" height="225">
</canvas>
```

Давайте добавим штриховую рамку, чтобы видеть, с чем мы имеем дело. Холст с рамкой (очерчивающей его границы) показан на рис. 4.1.

Рис. 4.1. "Холст" (элемент `<canvas>`) с рамкой

Холст с рамкой

На одной и той же странице можно иметь более одного элемента <canvas>. Каждый элемент <canvas> находит свое отражение в DOM, и каждый из них поддерживает собственный статус. Если вы присвоите каждому элементу <canvas> атрибут id, вы сможете получать к ним доступ по этим атрибутам, точно так же, как и к любым другим элементам.

Давайте расширим нашу стандартную разметку элемента canvas так, чтобы добавить атрибут id:

```
<canvas id="a" width="300" height="225"></canvas>
```

Теперь вы с легкостью сможете находить в DOM нужные элементы <canvas> по их идентификаторам

```
var a_canvas = document.getElementById("a");
```

Простейшие геометрические фигуры

В табл. 4.2 вкратце описана поддержка возможностей рисования на "холсте" простейших геометрических фигур в популярных современных браузерах и мобильных устройствах.

Таблица 4.2. *Поддержка возможностей рисования на "холсте" простейших геометрических фигур в популярных современных браузерах и мобильных устройствах*

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Для поддержки элемента <canvas> в Internet Explorer требуется библиотека стороннего разработчика explorercanvas.

Каждый элемент <canvas> изначально пуст. Это плохо, поэтому давайте для начала что-нибудь нарисуем.

Чтобы начать рисование, необходимо щелкнуть мышью по элементу <canvas>. Чтобы вызвать функцию, вычерчивающую прямоугольник (**http://diveintohtml5.org/canvas.html**), необходимо вызвать обработчик onclick. Этот обработчик вызывает функцию draw_b(), код которой приведен в листинге 4.1.

Листинг 4.1. Код функции draw_b(), вычерчивающей прямоугольник

```
function draw_b() {
  var b_canvas = document.getElementById("b");
  var b_context = b_canvas.getContext("2d");
  b_context.fillRect(50, 25, 150, 100);
}
```

Первая строка кода функции не представляет собой ничего особенного; она просто находит элемент <canvas> в DOM.

Затем выполняется следующая строка:

```
var b_context = b_canvas.getContext("2d");
```

Каждый элемент <canvas> имеет *контекст рисования* (drawing context), в котором и происходит все, представляющее интерес. Как только будет найден элемент <canvas> в составе DOM (используя document.getElementById() или любой другой метод по вашему усмотрению), вам следует вызвать его метод getContext(). Методу getContext() *необходимо* передать строку "2d".

Вопрос к Профессору Разметки

В: Бывают ли трехмерные элементы <canvas>?

О: Пока еще нет. Некоторые сторонние разработчики экспериментировали с API трехмерных элементов <canvas>, но ни один из этих API не был стандартизован. В примечаниях к спецификации HTML5 говорится о том, что последующие версии, возможно, будут определять 3d-контекст.

Итак, теперь у вас есть элемент <canvas>, и вы получили его контекст рисования. Именно в контексте рисования и определяются все методы и свойства рисования. Существует целая группа свойств и методов, посвященных вычерчиванию прямоугольников:

- Свойство fillStyle может быть цветом CSS, орнаментом или градиентом. (О градиентах речь пойдет чуть позже.) По умолчанию свойство fillStyle представляет собой сплошную черную заливку, но вы можете изменить его по собственному усмотрению на любой другой вариант. Каждый контекст рисования "помнит" собственные свойства до тех пор, пока страница остается открытой, если только вы сами не измените их.
- Функция fillRect(x, y, width, height) вычерчивает прямоугольник с использованием определенного на текущий момент стиля заливки.
- Свойство strokeStyle аналогично свойству fillStyle — это может быть цвет CSS, орнамент или градиент.
- strokeRect(x, y, width, height) вычерчивает прямоугольник, используя определенный на данный момент стиль линии. Функция strokeRect не выполняет заливки вычерченной фигуры, она просто обводит ее контур.
- clearRect(x, y, width, height) — эта функция выполняет очистку пикселов в пределах указанного прямоугольника.

Вопрос к Профессору Разметки

В: Можно ли выполнить "сброс" параметров элемента canvas?

О: Да. Переустановка параметров ширины (width) или высоты (height) элемента <canvas> приводит к стиранию его содержимого и сбросу всех параметров его контекста рисования, устанавливая для них значения, принятые по умолчанию. На прак-

тике, вам даже не нужно *менять значение* ширины; достаточно просто еще раз указать текущее ее значение, например, так:

```
var b_canvas = document.getElementById("b");
b_canvas.width = b_canvas.width;
```

Вернемся к примеру, иллюстрирующему вычерчивание прямоугольника, приведенному в листинге 4.1.

Вычерчивание прямоугольника выполняется, как показано в листинге 4.2.

Листинг 4.2. Фактическое вычерчивание прямоугольника

```
var b_canvas = document.getElementById("b");
var b_context = b_canvas.getContext("2d");
b_context.fillRect(50, 25, 150, 100);
```

Вызов метода `fillRect()` приводит к вычерчиванию прямоугольника и заполнению его заливкой, имеющей стиль, указанный на данный момент (по умолчанию это будет сплошная черная заливка, если вы не указали другого варианта, или до тех пор, пока вы этого не сделаете). Прямоугольник будет ограничен координатами его левого верхнего угла (50, 25), а также шириной (150), и высотой (100). Чтобы получить лучшее представление о том, как все это работает, давайте рассмотрим координатную систему элемента `<canvas>`.

Координатная система элемента <canvas>

Элемент `<canvas>` представляет собой плоскую сетку (рис. 4.2). Точка с координатами (0, 0) находится в левом верхнем углу элемента `<canvas>`. Вдоль оси X значения координат растут по направлению к правому краю элемента `<canvas>`. Вдоль оси Y значения координат возрастают в направлении от верхней границы к нижней.

Приведенная на иллюстрации координатная сетка была вычерчена при помощи элемента `<canvas>`. Она включает в свой состав следующие компоненты:

- набор серых вертикальных линий;
- набор серых горизонтальных линий;
- две черных горизонтальных линии;
- две небольших черных диагональных линии, которые формируют стрелку;
- две черных вертикальных линии;
- еще две небольших диагональных черных линии, которые формируют еще одну стрелку;
- букву "x";
- букву "y";
- текст "(0, 0)" рядом с верхним левым углом;
- текст "(500, 375)" рядом с правым нижним углом;
- точку в левом верхнем углу и еще одну — в правом нижнем углу.

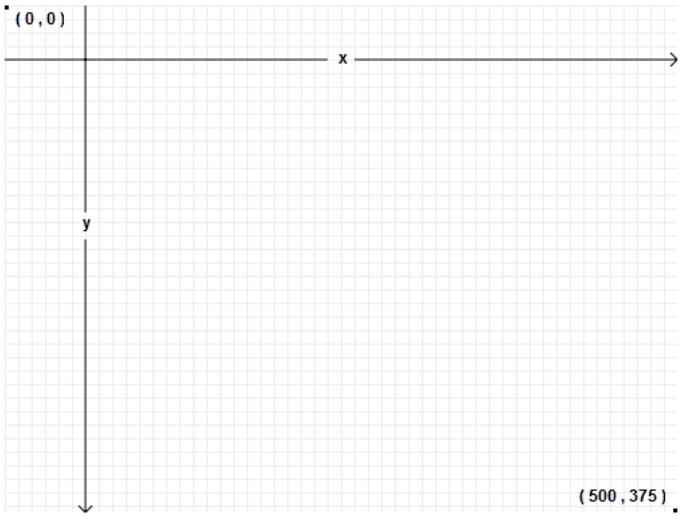


Рис. 4.2. Диаграмма координат элемента `<canvas>`

Во-первых, нам нужно определить сам элемент `<canvas>`. Элемент `<canvas>` определяет параметры `width` и `height`, а также `id`, так что мы сможем найти его впоследствии:

```
<canvas id="c" width="500" height="375"></canvas>
```

Затем нам необходим скрипт, чтобы найти элемент `<canvas>` в DOM и получить его контекст рисования. Этот скрипт представлен в листинге 4.3.

Листинг 4.3. Скрипт для поиска элемента `<canvas>` в DOM и получения его контекста рисования

```
var c_canvas = document.getElementById("c");
var context = c_canvas.getContext("2d");
```

Теперь можно начинать рисовать линии.

Вычерчивание линий

В табл. 4.3 вкратце описана поддержка возможностей вычерчивания линий на "холсте" в популярных современных браузерах и мобильных устройствах.

***Таблица 4.3.** Поддержка возможностей по вычерчиванию линий на "холсте" в популярных браузерах и мобильных устройствах*

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Для поддержки `<canvas>` в Internet Explorer требуется библиотека стороннего разработчика `explorercanvas`.

Представьте себе, что вы рисуете картину тушью. Но вы не хотите сразу же взять и начинать рисовать тушью, потому что так можно допустить ошибку. Вместо этого вы предпочли бы начертить линии и фигуры карандашом, а когда вы будете довольны полученным результатом, его можно будет и обвести.

Каждый элемент `<canvas>` имеет *путь* или траекторию (path). Определение пути напоминает рисование карандашом. Вы можете нарисовать все, что вам захочется, но это не станет частью законченной работы, до тех пор, пока вы не возьмете перо и не обведете тушью вычерченную траекторию.

Чтобы вычертить прямые линии "карандашом", вы можете воспользоваться следующими двумя методами:

- `moveTo(x, y)` — этот метод перемещает карандаш в указанную начальную точку.
- `lineTo(x, y)` — этот метод вычерчивает линию до указанной конечной точки.

Чем больше вы вызываете метод `moveTo()` и метод `lineTo()`, тем длиннее становится путь. Это — "карандашные" методы, и вы можете вызывать их так часто, как можете и хотите, но вы не увидите ничего на своем "холсте" до тех пор, пока не вызовете один из методов "обводки".

Начнем рисовать с серых линий сетки. Техника рисования продемонстрирована в листинге 4.4.

Листинг 4.4. Вычерчивание вертикальных серых линий сетки

```
for (var x = 0.5; x < 500; x += 10) {
  context.moveTo(x, 0);
  context.lineTo(x, 375);
}
```

Теперь нарисуем горизонтальные линии (листинг 4.5).

Листинг 4.5. Вычерчивание горизонтальных линий сетки

```
for (var y = 0.5; y < 375; y += 10) {
  context.moveTo(0, y);
  context.lineTo(500, y);
}
```

Все это были "карандашные" методы. На самом деле, на "холсте" еще ничего не было нарисовано. Чтобы сделать внесенное изменение постоянным, нам нужно "обвести" вычерченные линии, как показано в листинге 4.6.

Листинг 4.6. "Обводка" вычерченных линий сетки

```
context.strokeStyle = "#eee";
context.stroke();
```

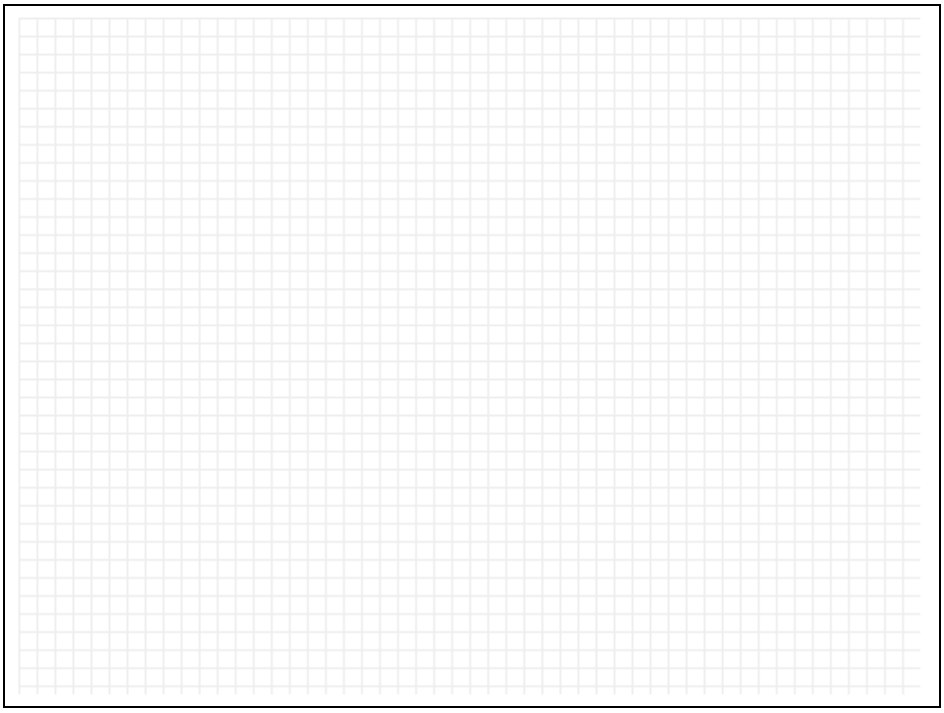


Рис. 4.3. Сетка, нарисованная на холсте

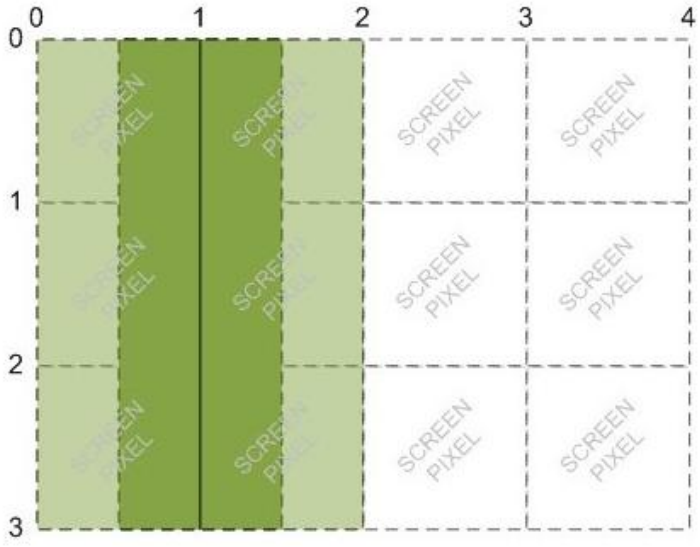


Рис. 4.4. Возможность отобразить половину пиксела отсутствует, поэтому толщина получившейся линии составит два пиксела

`stroke()` — это один из методов, работающих с "тушью". Он берет сложный путь, который вы определили с помощью всех этих вызовов `moveTo()` и `lineTo()`, и фактически вычерчивает эту траекторию на холсте. Метод `strokeStyle` управляет цветом линий. Полученный результат представлен на рис. 4.3.

Вопрос к Профессору Разметки

В: Почему вы присвоили `x` и `y` начальные значения 0.5? Почему бы не начать с 0?

О: Представьте себе каждый пиксел в виде большого квадрата. Целочисленные координаты (0, 1, 2…) представляют собой координаты вершин этих квадратов. Если вы вычертите линию толщиной один пиксел между точками, имеющими целочисленные координаты, она перекроет противоположные стороны пиксельного квадрата, и в результате вы получите линию, толщина которой составляет не один, а два пиксела. Чтобы вычертить линию толщиной только один пиксел, вам нужно сдвинуть координаты на 0.5 в направлении, перпендикулярном линии.

Например, если вы попытаетесь провести линию, соединяющую точки с координатами (1, 0) и (1, 3), то браузер проведет линию, закрывающую по 0.5 экранного пиксела дополнительно с каждой стороны линии `x=1`. Возможность отобразить половину пиксела отсутствует, поэтому толщина получившейся линии составит два пиксела (рис. 4.4).

Но если вы проведете линию, соединяющую точки (1.5, 0) и (1.5, 3), тогда браузер проведет линию толщиной 1 пиксел, закрывающую по 0.5 экранных пиксела с каждой стороны от линии `x=1.5`, в результате чего толщина этой линии составит как раз ровно 1 пиксел (рис. 4.5).

Данные диаграммы были предоставлены Джейсоном Джонсоном (Jason Johnson).

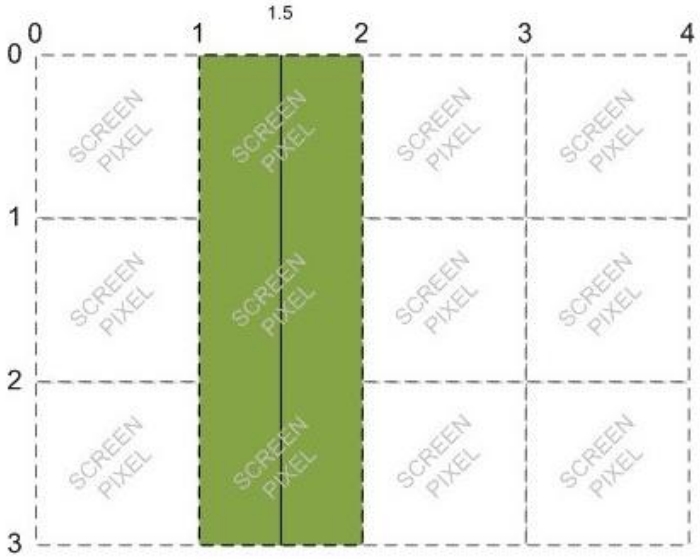


Рис. 4.5. Но если вы проведете линию, соединяющую точки (1.5, 0) и (1.5, 3), тогда браузер проведет линию толщиной 1 пиксел

Теперь вычертим горизонтальную стрелку. Все линии и кривые, входящие в состав траектории (path), имеют один и тот же цвет (или градиент — к обсуждению этой темы мы вскоре приступим). Нам нужно вычертить стрелку другим цветом — черным, а не серым, поэтому нам требуется начать новую траекторию.

Новая траектория выглядит так, как показано в листинге 4.7.

Листинг 4.7. Вычерчивание горизонтальной стрелки

```
context.beginPath();
context.moveTo(0, 40);
context.lineTo(240, 40);
context.moveTo(260, 40);
context.lineTo(500, 40);
context.moveTo(495, 35);
context.lineTo(500, 40);
context.lineTo(495, 45);
```

Вертикальная стрелка выглядит во многом аналогично. Поскольку вертикальная стрелка имеет тот же цвет, что и горизонтальная, нам не требуется начинать еще одну новую траекторию. Две стрелки будут представлять собой части одной и той же траектории. Вычерчивание вертикальной стрелки показано в листинге 4.7. Обратите внимание, что в листинге 4.8 мы не начинаем работу с создания новой траектории, потому что в этом нет необходимости.

Листинг 4.8. Вычерчивание вертикальной стрелки

```
context.moveTo(60, 0);
context.lineTo(60, 153);
context.moveTo(60, 173);
context.lineTo(60, 375);
context.moveTo(65, 370);
context.lineTo(60, 375);
context.lineTo(55, 370);
```

Я уже сказал о том, что эти стрелки должны иметь черный цвет, но свойство `strokeStyle` пока все еще имеет прежнее значение (серый цвет). Свойства `fillStyle` и `strokeStyle` не сбрасываются при начале вычерчивания новой траектории. Это нормально, потому что мы воспользовались лишь последовательностью "карандашных" методов. Но перед тем, как фактически "обвести" карандашные линии "тушью", нам нужно установить для свойства `strokeStyle` черный цвет, в противном случае мы будем рисовать серым по серому и вряд ли сможем увидеть результат нашей работы! Строки кода, приведенные в листинге 4.9, изменят цвет линий на черный, а затем выполнят их фактическое вычерчивание в области "холста".

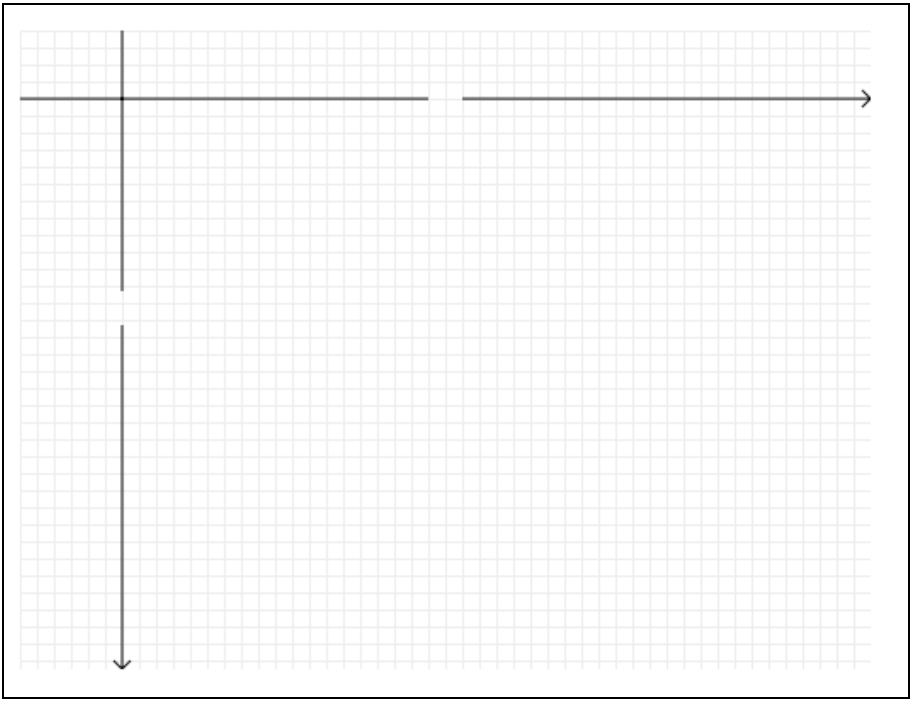


Рис. 4.6. Оси без меток, вычерченные на холсте

Листинг 4.9. Изменение цвета линий на черный и фактическое их вычерчивание

```
context.strokeStyle = "#000";
context.stroke();
```

Результат будет выглядеть так, как показано на рис. 4.6.

Работа с текстом

В табл. 4.4 вкратце описана поддержка возможностей работы с текстом на "холсте" в популярных современных браузерах и мобильных устройствах.

Таблица 4.4. *Поддержка возможностей работы с текстом на "холсте" в популярных современных браузерах и мобильных устройствах*

IE*	Firefox†	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Для поддержки `<canvas>` в Internet Explorer требуется библиотека стороннего разработчика `explorercanvas`.

† Для поддержки Mozilla Firefox 3.0 требуется библиотека совместимости (compatibility shim).

В дополнение к рисованию линий на "холсте", вам требуется и возможность добавления текста. В отличие от текста на окружающей иллюстрацию части Web-страницы, здесь нет модели "поля" или "рамки". Это значит, что вам не будут доступны никакие из знакомых методов форматирования CSS: ни обтекания (float), ни границы (margins), ни дополнения пустых позиций (padding), ни переносы слов (word wrapping). (Хотя возможно, что вы даже думаете, что это хорошо!) Вы сможете задать некоторые атрибуты шрифтов, затем вам понадобится выбрать точку на "холсте" и начать с нее ввод вашего текста.

В контексте рисования вам будут доступны следующие атрибуты шрифтов:

- `font` — это может быть что угодно из того, что вы можете пожелать включить в правило CSS `font`. Сюда входят стиль шрифта (font style), вариант шрифта (font variant), толщина шрифта (font weight), размер шрифта (font size), высота линии (line height) и семейство шрифтов (font family).
- `textAlign` управляет выравниванием текста. Это свойство аналогично правилу CSS `text-align` (но не идентично ему). Возможны следующие значения: `start`, `end`, `left`, `right` и `center`.
- `textBaseline` — этот атрибут управляет местоположением ввода текста по отношению к начальной позиции. Возможны следующие значения: `top`, `hanging`, `middle`, `alphabetic`, `ideographic` и `bottom`.

`textBaseline` — это довольно "коварное" свойство (к тексту на английском языке это не относится, но вы можете вводить на "холст" любые символы в кодировке Unicode, а Unicode — это "коварная" кодировка). Спецификация HTML5 дает следующее пояснение, касающееся "базовых линий" текста (**http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#dom-context-2d-textbaseline**):

> "Верхняя часть кегельной площадки (em square)[1] находится примерно на уровне верхних контуров глифов шрифта, "висячая" базовая линия (hanging baseline) находится на том уровне, где закрепляются глифы наподобие крайнего справа (рис. 4.7), середина находится между верхней и нижней границами кегельной площадки, базовая линия алфавита расположена там, где закреплены нижние контуры таких символов, как Á, ÿ, f и Ω, идеографическая базовая линия (ideographic baseline) находится в месте закрепления таких глифов, как 私 и 達, а нижняя граница кегельной площадки находится примерно там, где расположена нижняя линия, ограничивающая нижние части глифов шрифта. Верхняя и нижняя линии ограничивающего блока могут быть достаточно далеко от этих базовых линий, потому что глифы могут выходить за рамки кегельной площадки".

[1] В металлическом наборе — верхняя прямоугольная или квадратная часть ножки литеры, на которой расположено выпуклое (печатающее) изображение буквы или другого знака. Края кегельной площадки называются габаритами литеры. Ее высота, выраженная в типографских пунктах, называется кеглем, а ширина — толщиной (total width) литеры. Боковые габариты также называются стенками литеры. Расстояние по горизонтали от стенки литеры до ближайшей точки на контуре знака называется полуапрошем. В металлическом шрифте величина кегельной площадки определяла все измерения литеры в наборе и верстке. В цифровом шрифте кегельная площадка важна только при проектировании шрифта как прямоугольник, в который вписывается изображение знака. См. **http://www.paratype.ru/help/term/terms.asp?code=41**, **http://www.weblibrary.biz/css/font/parametry_font**, **https://secure.wikimedia.org/wikipedia/en/wiki/Em_%28typography%29**. — *Прим. перев.*

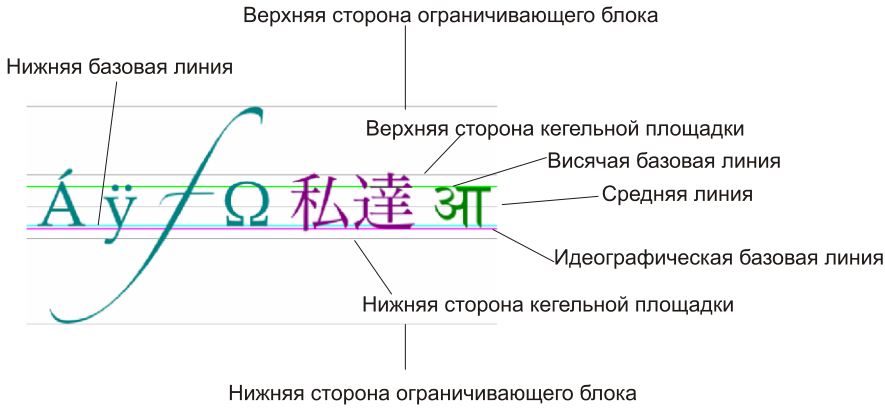


Рис. 4.7. Основные базовые линии текста

Для простых алфавитов наподобие английского, вы можете просто принять имеющиеся значения параметров `top`, `middle` или `bottom` для свойства `textBaseline`.

Давайте начнем вводить текст! Текст, введенный в пределах элемента `<canvas>`, наследует стиль и размер шрифта, установленные для элемента `<canvas>` как такового, но вы можете отменить это назначение, указав значение свойства `font` для контекста рисования, как показано в листинге 4.10.

Листинг 4.10. Установка значения свойства `font` для контекста рисования

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

Метод `fillText()` вычерчивает сам текст, как показано в листинге 4.11.

Листинг 4.11. Вычерчивание текста

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

Вопрос к Профессору Разметки

В: Можно ли использовать относительные размеры шрифта при вводе текста на элемент `<canvas>`?

О: Да. Как и любой другой элемент HTML на вашей странице, сам по себе элемент `<canvas>` имеет вычисляемые размеры шрифтов, которые получаются на основании правил CSS для вашей страницы. Если вы зададите свойство `context.font`, установив размер шрифта наподобие `1.5em` или `150%`, то ваш браузер умножит на это значение вычисленный размер шрифта для элемента `<canvas>`.

Допустим, что для текста в верхнем левом углу я хочу, чтобы верхняя линия шрифта проходила на уровне `y=5`. Но я ленив, и я не хочу измерять высоту текста и вычислять расположение базовой линии. Вместо этого я могу установить `textBaseline` на `top` и передать координату верхнего левого угла текстового поля, как показано в листинге 4.12.

Листинг 4.12. Изменение местоположения шрифта

```
context.textBaseline = "top";
context.fillText("( 0 , 0 )", 8, 5);
```

Теперь перейдем к правому нижнему углу. Допустим, что я хочу, чтобы правый нижний угол текста имел координаты (492,370) — как раз на расстоянии нескольких пикселов от правого нижнего угла "холста" — но при этом я не хочу сам замерять ширину или высоту текста. Я могу установить `textAlign` на `right` и `textBaseline` на `bottom`, а затем вызвать `fillText()`, передав методу координаты правого нижнего угла рамки, заключающей в себе текст, как показано в листинге 4.13.

Листинг 4.13. Указание координат правого нижнего угла текстового блока

```
context.textAlign = "right";
context.textBaseline = "bottom";
context.fillText("( 500 , 375 )", 492, 370);
```

Результат будет таким, как показано на рис. 4.8.

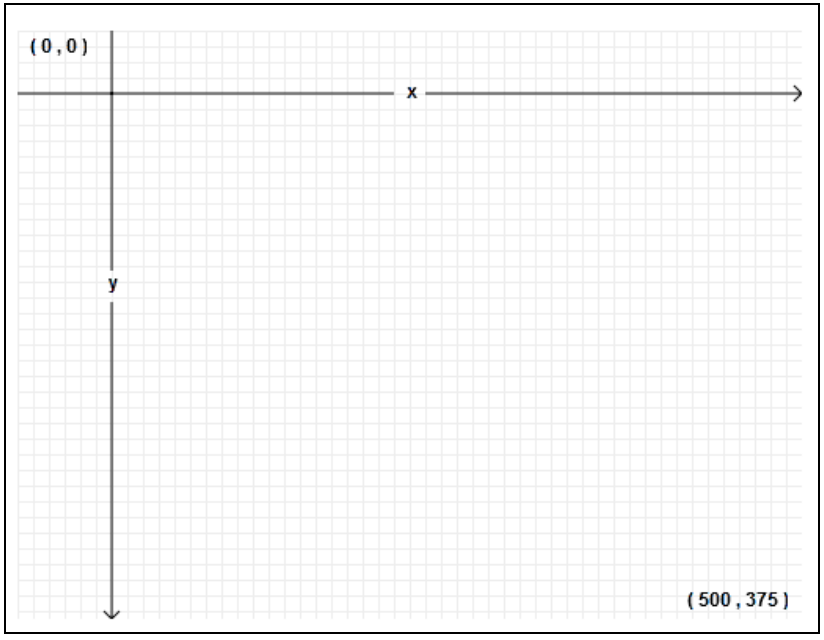


Рис. 4.8. Помеченные оси на "холсте"

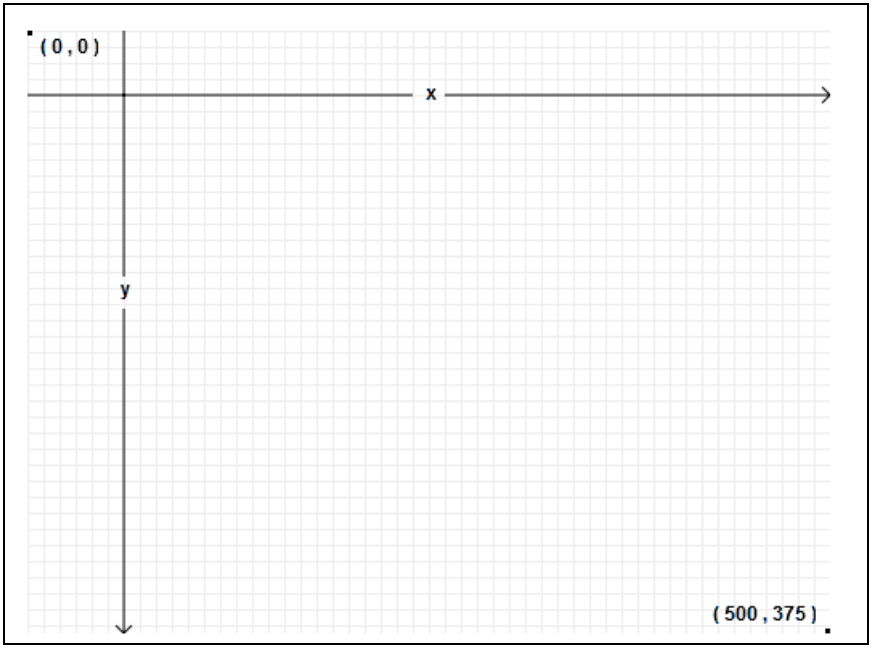


Рис. 4.9. Координатная диаграмма холста на холсте

Оп-ля! Мы забыли про угловые точки. О вычерчивании окружностей мы поговорим позднее. На данный момент, я слегка "схалтурю" и вычерчу вместо них прямоугольники, как показано в листинге 4.14.

Листинг 4.14. Вычерчивание двух "точек"

```
context.fillRect(0, 0, 3, 3);
context.fillRect(497, 372, 3, 3);
```

И это все! Вот окончательный результат (рис. 4.9).

Градиенты

В табл. 4.5 вкратце описана поддержка возможностей работы с градиентами на "холсте" в популярных современных браузерах и мобильных устройствах.

Таблица 4.5. *Поддержка возможностей работы с градиентами на "холсте" в популярных современных браузерах и мобильных устройствах*

	IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
Линейные градиенты	7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
Радиальные градиенты	—	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Для поддержки `<canvas>` в Internet Explorer требуется библиотека стороннего разработчика explorercanvas.

Ранее в этой главе я рассказывал вам о том, как вычертить прямоугольник, закрашенный сплошным цветом, затем — как вычертить линию, залитую сплошным цветом. Но геометрические фигуры и линии не ограничиваются использованием сплошных заливок. При их вычерчивании вы можете применять все "трюки" и "фокусы" с градиентами. Давайте рассмотрим пример.

В данном случае разметка выглядит точно так же, как и для любого другого элемента `<canvas>`.

```
<canvas id="d" width="300" height="225"></canvas>
```

Сначала нам требуется найти сам элемент `<canvas>` и его контекст рисования:

```
var d_canvas = document.getElementById("d");
var context = d_canvas.getContext("2d");
```

Как только будет получен контекст рисования, мы можем приступать к определению градиента. Градиент представляет собой плавный переход между двумя или несколькими цветами. Контекст рисования элемента `<canvas>` поддерживает два типа градиентов (gradients):

- `createLinearGradient(x0, y0, x1, y1)` — выполняет заливку вдоль линии, проведенной между точками с координатами `(x0, y0)` и `(x1, y1)`.
- `createRadialGradient(x0, y0, r0, x1, y1, r1)` — выполняет заливку конуса между двумя окружностями. Первые три параметра задают начальную окружность с центром в точке `(x0, y0)` и радиусом `r0`. Последние три параметра задают конечную окружность, с центром в точке `(x1, y1)` и радиусом `r1`.

Для начала создадим линейный градиент. Градиенты могут иметь любую ширину, но я в данном примере создам градиент с шириной 300 пикселов, как и у нашего элемента `<canvas>`.

```
var my_gradient = context.createLinearGradient(0, 0, 300, 0);
```

Так как значения ординаты `(y)` — это второй и четвертый параметры — оба равны нулю, этот градиент будет равномерно меняться в направлении слева направо.

Получив объект-градиент, мы можем определить его цвета. Градиент имеет две или большее количество *цветовых остановок* (color stops). Цветовые остановки могут располагаться где угодно на градиентной шкале. Чтобы добавить цветовую остановку, вы должны указать ее позицию на градиентной шкале. Градиентные позиции могут находиться где угодно в пределах интервала от 0 до 1.

Определим градиент для цветовой растяжки от черного к белому, как показано в листинге 4.15.

Листинг 4.15. Определение градиента для цветовой растяжки от черного к белому

```
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
```

Само по себе определение градиента не приводит к вычерчиванию чего-либо на "холсте". Определение градиента — это просто объект, который хранится где-то

в памяти. Чтобы изобразить градиент, вам нужно установить соответствующее значение (gradient) для свойства `fillStyle` и вычертить геометрическую фигуру, обладающую этим свойством (например, линию или прямоугольник), как показано в листинге 4.16.

Листинг 4.16. Установка стиля заливки на градиент

```
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Результат будет выглядеть так, как показано на рис. 4.10.

Предположим, что вам требуется, чтобы градиентные переходы осуществлялись сверху вниз. Когда вы создаете объект-градиент, оставьте постоянными значения `x` (первый и третий параметры метода) и укажите значения `y` (второй и четвертый параметры) из диапазона от 0 до высоты объекта `<canvas>`, как показано в листинге 4.17.

Листинг 4.17. Создание вертикального градиента

```
var my_gradient = context.createLinearGradient(0, 0, 0, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Результат будет выглядеть так, как показано на рис. 4.11.

Кроме того, вы можете указывать и диагональные градиенты.

В этом случае изменяются обе координаты — и x, и y, как показано в листинге 4.18.

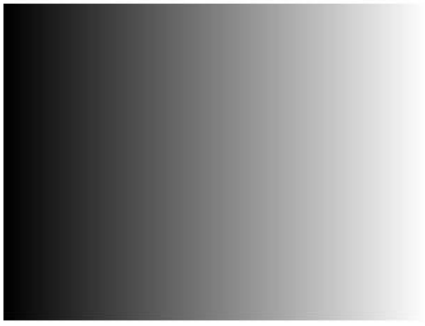


Рис. 4.10. Линейный градиент с растяжкой слева направо

Рис. 4.11. Вертикальный градиент

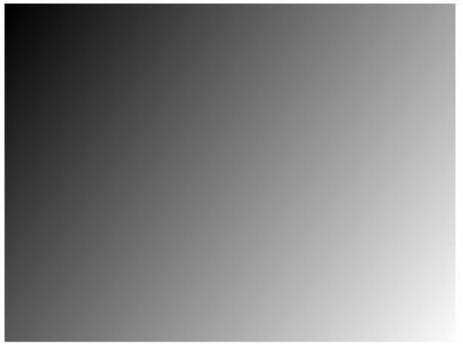


Рис. 4.12. Диагональный градиент

Листинг 4.18. Создание диагонального градиента

```
var my_gradient = context.createLinearGradient(0, 0, 300, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Результат будет выглядеть следующим образом (рис. 4.12).

Изображения

В табл. 4.6 вкратце описана поддержка возможностей работы с изображениями на "холсте" в популярных современных браузерах и мобильных устройствах.

Таблица 4.6. *Поддержка возможностей работы с изображениями на "холсте" в популярных современных браузерах и мобильных устройствах*

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Для поддержки `<canvas>` в Internet Explorer требуется библиотека стороннего разработчика `explorercanvas`.

На рис. 4.13 представлено изображение кошки с помощью элемента `<img>`. Изображение этой же кошки с помощью элемента `<canvas>` будет выглядеть точно так же.

Контекст рисования элемента `<canvas>` определяет метод `drawImage()`, с помощью которого и осуществляется рисование картинки на "холсте". Метод может принимать три, пять или девять аргументов.

- `drawImage(image, dx, dy)` — метод принимает в качестве аргумента изображение и рисует его на "холсте". Указанные координаты `(dx, dy)` становятся ко-

ординатами верхнего левого угла изображения. Если указать координаты `(0, 0)`, то верхний левый угол изображения совпадет с верхним левым углом "холста".

Рис. 4.13. Изображение кошки, созданное с помощью элемента `<img>`

- `drawImage(image, dx, dy, dw, dh)` — метод принимает в качестве аргумента изображение, масштабирует его по ширине `dw` и высоте `dh` и выводит изображение в области элемента `<canvas>` так, чтобы верхний левый угол этого изображения находился в точке с координатами `(dx, dy)`.
- `drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)` — принимает изображение в качестве аргумента, обрезает его по границам прямоугольника (`sx`, `sy`, `sw`, `sh`), масштабирует его по размерам `(dw, dh)`, и выводит в области элемента, начиная с точки, имеющей координаты `(dx, dy)`.

Спецификация HTML5 объясняет параметры `drawImage()` следующим образом (**http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#images**):

"Исходный прямоугольник — это прямоугольник, расположенный в пределах исходного изображения, вершины которого расположены в точках с координатами `(sx, sy)`, `(sx+sw, sy)`, `(sx+sw, sy+sh)`, `(sx, sy+sh)`.

Результирующий прямоугольник — это прямоугольник, расположенный в пределах элемента canvas, вершины которого находятся в точках `(dx, dy)`, `(dx+dw, dy)`, `(dx+dw, dy+dh)`, `(dx, dy+dh)`".

Отображение исходного изображения на "холсте" показано на рис. 4.14.

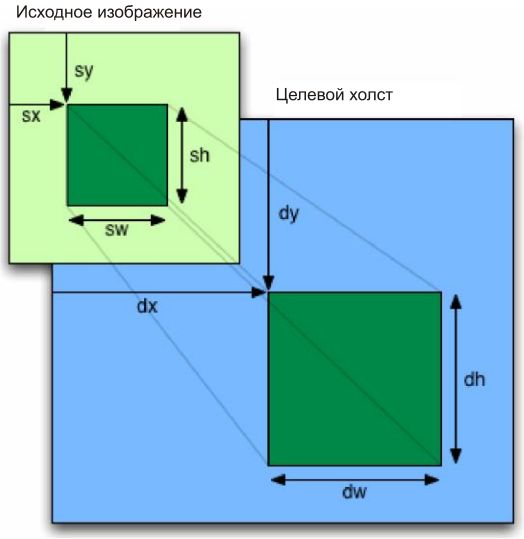


Рис. 4.14. Способ, которым `drawImage()` переносит изображение на холст

Чтобы вывести изображение на элемент `<canvas>`, вам для начала необходимо само это изображение. Оно может быть получено из существующего элемента `<img>`, или же вы можете самостоятельно создать элемент `Image()` с помощью JavaScript. В любом случае вам необходимо гарантировать, что изображение полностью загрузилось, прежде чем вы сможете начать его вывод на элемент `<canvas>`.

Если вы пользуетесь существующим элементом `<img>`, вы спокойно можете отрисовывать его на "холсте", в течение всего времени обработки события `window.onload`.

Использование существующего элемента `<img>` показано в листинге 4.19.

Листинг 4.19. Вывод изображений на "холст" из элемента <img>

```
<img id="cat" src="images/cat.png" alt="sleeping cat" width="177" height="113">
<canvas id="e" width="177" height="113"></canvas>
<script>
window.onload = function() {
  var canvas = document.getElementById("e");
  var context = canvas.getContext("2d");
  var cat = document.getElementById("cat");
  context.drawImage(cat, 0, 0);
};
</script>
```

Если вы самостоятельно создаете объект-изображение с помощью JavaScript, вы можете выполнять вычерчивание изображения на "холсте" в течение времени обработки события `Image.onload`.

Использование объекта `Image()` продемонстрировано в листинге 4.20.

Листинг 4.20. Использование объекта Image() для вывода изображений на "холст"

```
<canvas id="e" width="177" height="113"></canvas>
<script>
  var canvas = document.getElementById("e");
  var context = canvas.getContext("2d");
  var cat = new Image();
  cat.src = "images/cat.png";
  cat.onload = function() {
    context.drawImage(cat, 0, 0);
  };
</script>
```

Не обязательные для использования третий и четвертый параметры метода `drawImage()` управляют масштабированием изображения. В рассматриваемом при-

мере мы имеем дело с тем же самым изображением, но уменьшенным вдвое по обеим осям и выводимым последовательно в начальных точках с координатами левой верхней вершины, расположенной в пределах одного и того же элемента <canvas>.

Скрипт, выполняющий "клонирование кошек", представлен в листинге 4.21.

Листинг 4.21. Кошки размножаются клонированием!

```
cat.onload = function() {
  for (var x = 0, y = 0;
       x < 500 && y < 375;
       x += 50, y += 37) {
    context.drawImage(cat, x, y, 88, 56);
  }
};
```

Результат работы этого скрипта представлен на рис. 4.15.

Все только что сказанное приводит к закономерному вопросу: во-первых, зачем вообще нам нужно выводить изображения на "холст"? Какое преимущество дает вам эта избыточная сложность "рисования картинки на холсте" по сравнению с обычным элементом <img> и некоторыми правилами CSS? Ведь даже наш пример с "клонированием кошек" можно воспроизвести при помощи 10 перекрывающихся элементов <img>.

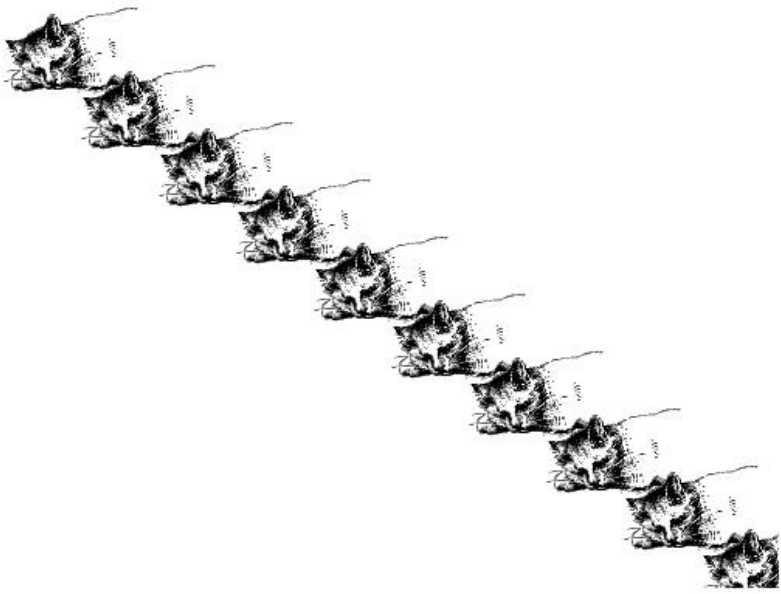


Рис. 4.15. Клонирование кошек!

Простой ответ можно сформулировать так: по некоторым причинам, вам может потребоваться рисование текста на "холсте". Координатная диаграмма "холста" включала текст, линии и геометрические фигуры; текст, выведенный на "холст", представляет собой лишь часть более сложной работы. Более сложная диаграмма может использовать метод `drawImage()`, чтобы включать значки (icons), спрайты (sprites) или более сложную графику.

Как быть с IE?

Microsoft Internet Explorer (вплоть до новейшей на момент написания книги версии 8, включительно) не поддерживает canvas API, хотя Internet Explorer 9 поддерживает canvas API в полном объеме (**http://msdn.microsoft.com/en-us/ie/ff468705.aspx#_HTML5_canvas**). Однако более ранние версии Internet Explorer поддерживают патентованную технологию Microsoft, известную под названием VML (**http://msdn.microsoft.com/en-us/library/bb250524%28v=vs.85%29.aspx**), которая позволяет выполнять большинство задач, возложенных на элемент `<canvas>`. Таким образом, на свет появился скрипт `excanvas.js` .

`Explorercanvas` (`excanvas.js`) представляет собой библиотеку JavaScript на основе открытого кода, лицензированную Apache. Эта библиотека реализует canvas API в Internet Explorer. Чтобы воспользоваться ею, включите в начало страницы элемент `<script>`. Это делается так, как показано в листинге 4.22.

Листинг 4.22. Использование `excanvas.js` в версиях Internet Explorer, более ранних, чем IE9

```
<html>
<head>
  <meta charset="utf-8">
  <title>Dive Into HTML5</title>
    <!--[if lt IE 9]>
    <script src="excanvas.js"></script>
    <![endif]-->
 </head>
<body>
  ...
</body>
</html>
```

Фрагменты `<!--[if lt IE 9]>` и `<![endif]-->` представляют собой условные комментарии. Internet Explorer интерпретирует их как утверждение `if`: "если используемый на данный момент браузер — одна из версий Internet Explorer, более ранних, чем 9, то необходимо выполнить данный блок". Любой другой браузер будет трактовать этот блок как комментарий HTML. В результате этого Internet Explorer загрузит скрипт `excanvas.js` и выполнит его, а остальные браузеры его

полностью проигнорируют (не загрузят, не выполнят, и вообще не предпримут никаких действий). За счет этого данная страница загрузится быстрее в тех браузерах, которые обеспечивают встроенную поддержку canvas API.

Как только вы включите скрипт `excanvas.js` в элемент `<head>` вашей страницы, вам больше не потребуется предпринимать никаких других усилий для обеспечения поддержки элемента `<canvas>` в Internet Explorer. Достаточно будет просто включать элементы `<canvas>` в вашу разметку или создавать их динамически при помощи JavaScript. Следуя инструкциям, приведенным в данной главе, вы сможете получать контекст рисования элемента `<canvas>`, и после этого можно будет рисовать геометрические фигуры, вводить текст и пользоваться орнаментами (patterns).

Это хорошо, но это еще не все. Существует несколько ограничений:

- Градиенты могут быть только линейными. Радиальные градиенты (**https://developer.mozilla.org/En/Canvas_tutorial/Applying_styles_and_colors#A_createRadialGradient_example**) не поддерживаются.
- Орнаменты должны повторяться в обоих направлениях.
- Области обрезки (**http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#clipping-region**) не поддерживаются.
- Непропорциональное масштабирование (**http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#dom-context-2d-scale**) некорректно обрабатывает контуры.
- Все это работает медленно. Это не должно вас шокировать, во-первых, потому что модуль интерпретации JavaScript в Internet Explorer работает медленнее, чем у других браузеров. Как только вы начнете рисовать сложные фигуры с помощью библиотеки JavaScript, которая транслирует команды, используя совершенно другую технологию, ситуация осложнится. Вы не заметите падения производительности при работе с простыми примерами, скажем, такими как вычерчивание нескольких линий или трансформация изображения, но вы сразу столкнетесь с проблемами, если начнете экспериментировать с анимацией на "холстах" или другими сложными задачами.

Использование `excanvas.js` имеет еще один "подводный камень", и именно с этой проблемой я столкнулся при разработке примеров для этой главы. `ExplorerCanvas` автоматически инициализирует собственный интерфейс-подмену для элемента `<canvas>` всякий раз, когда вы включаете скрипт `excanvas.js` в вашу HTML-страницу. Но это не значит, что Internet Explorer будет готов к его использованию немедленно. При определенных обстоятельствах вы можете попасть в такую ситуацию, когда интерфейс-подмена будет *почти*, но еще не полностью готов к использованию. Основным симптомом наступления этой ситуации является то, что Internet Explorer выводит сообщение, говорящее о том, что "object doesn't support this property or method" ("объект не поддерживает этот метод или свойство"), когда вы пытаетесь сделать хоть что-нибудь с элементом `<canvas>`, например, получить его контекст рисования.

Простейшее решение проблемы заключается в том, чтобы отложить все манипуляции с элементом `<canvas>` до тех пор, пока не сработает событие `onload`. Это время ожидания может оказаться довольно продолжительным, если ваша страница

содержит большое количество изображений или видео, то это отсрочить срабатывание события `onload`, но зато это даст время библиотеке `ExplorerCanvas` реализовать свои функции.

"Живой" пример

Halma (**https://secure.wikimedia.org/wikipedia/en/wiki/Halma**) — это старинная настольная игра на расчерченной доске. Существует множество ее вариаций. В рассматриваемом примере я создал версию Halma для одного игрока, в которой используется 9 фишек на доске 9×9. В начале игры фишки образуют квадрат 3×3 в левом нижнем углу игровой доски. Целью игры является перемещение всех фишек так, чтобы они выстроились в виде квадрата 3×3 в правом верхнем углу игровой доски, выполнив при этом минимальное количество ходов.

В игре Halma существуют два типа допустимых ходов:

- Перемещение фишки на любое из примыкающих к ней пустых полей. "Пустым" считается поле, которое в данный момент не занято другой фишкой. "Примыкающим" считается поле, расположенное непосредственно к северу, югу, востоку, западу, северо-востоку, северо-западу, юго-востоку или юго-западу от текущего поля. Доску нельзя переворачивать с одной стороны на другую. Если фишка находится в крайнем левом столбце, ее нельзя переместить на запад, северо-запад или юго-запад. Если фишка находится в нижнем ряду, ее нельзя переместить на юг, юго-восток или юго-запад.
- Фишку можно переместить через другую фишку, расположенную на примыкающем к ней поле, причем за один ход можно выполнить несколько таких движений. Это значит, что, если вы переместили фишку через фишку, расположенную рядом, и рядом с полем, на которое она была перемещена, есть *еще одна* фишка, то вы можете "перепрыгнуть" и через нее тоже, и это движение будет считаться одним ходом. Фактически, вы можете "перескочить" через любое количество таких фишек, и все эти перемещения будут засчитаны за один ход. Так как целью игры является минимизация количества ходов, то правильной стратегией в Halma является построение комбинаций с последующим использованием многошаговых последовательностей, позволяющих выполнить за один ход как можно большее количество результативных переходов.

Вот сама игра (рис. 4.16). Вы можете также самостоятельно решить эту задачку, загрузив игру на отдельную страницу и сыграв в нее (**http://diveintohtml5.org/examples/canvas-halma.html**), постаравшись добиться наилучшего результата. Кроме того, вы можете "поковырять" ее, пользуясь инструментарием разработчика, предоставляемого вашим браузером.

Как все это работает? Если задали этот вопрос, я очень рад. В книге я не могу привести код игры полностью, но вы можете скачать его по следующему адресу: **http://diveintohtml5.org/examples/halma.js**. В данном же тексте я пропущу большую часть кода игры, как таковой, и сосредоточу ваше внимание на тех фрагментах, которые "ответственны" за взаимодействие с "холстом" и за реагирование на щелчки мышью в его области.

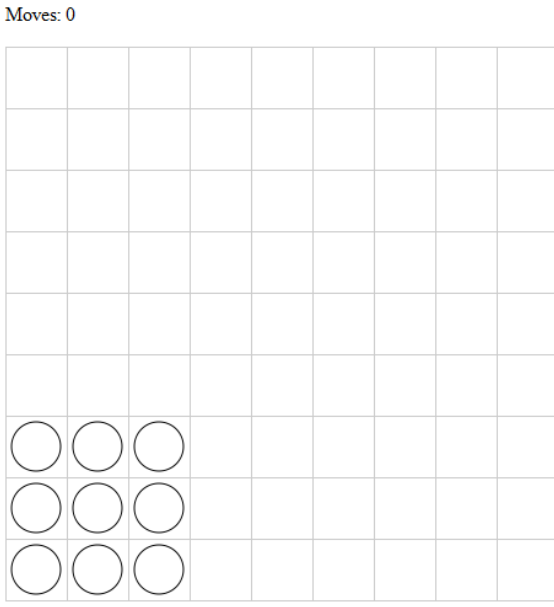


Рис. 4.16. Начальная позиция игры Halma

Во время загрузки страницы мы инициализируем игру, устанавливая размеры самого элемента `<canvas>` и сохраняя ссылку на его контекст рисования, как показано в листинге 4.23.

Листинг 4.23. Инициализация игры Halma во время загрузки страницы

```
gCanvasElement.width = kPixelWidth;
gCanvasElement.height = kPixelHeight;
gDrawingContext = gCanvasElement.getContext("2d");
```

После этого нам необходимо проделать нечто новое, такое, чего до сих пор мы еще ни разу не делали: нам требуется добавить сервис "прослушивания" событий к элементу `<canvas>`, чтобы отслеживать события типа "щелчок мышью":

```
gCanvasElement.addEventListener("click", halmaOnClick, false);
```

Функция `halmaOnClick()` вызывается, когда пользователь щелкает мышью где-либо в пределах "холста". Ее аргументом является объект `MouseEvent`, содержащий информацию о том, где именно был выполнен щелчок мышью (листинг 4.24).

Листинг 4.24. Функция `halmaOnClick()`

```
function halmaOnClick(e) {
    var cell = getCursorPosition(e);

    // the rest of this is just gameplay logic
```

```
    for (var i = 0; i < gNumPieces; i++) {
if ((gPieces[i].row == cell.row) &&
            (gPieces[i].column == cell.column)) {
            clickOnPiece(i);
            return;
        }
    }
    clickOnEmptyCell(cell);
}
```

Следующий шаг заключается в том, чтобы принять объект `MouseEvent` на обработку и вычислить квадрат доски (поле), в котором был выполнен щелчок мышь. Игровая доска Halma занимает всю область элемента `<canvas>`, поэтому любой щелчок мышью, так или иначе, выполняется по одному из полей. Нам требуется только определить — где именно (листинг 4.25). Это сложно, потому что практически каждый браузер по-своему реализует обработку щелчков мышью.

Листинг 4.25. Вычисление квадрата доски, по которому был выполнен щелчок мышью

```
function getCursorPosition(e) {
    var x;
    var y;
    if (e.pageX != undefined && e.pageY != undefined) {
       x = e.pageX;
       y = e.pageY;
    }
    else {
       x = e.clientX + document.body.scrollLeft +
            document.documentElement.scrollLeft;
       y = e.clientY + document.body.scrollTop +
            document.documentElement.scrollTop;
    }
```

Дойдя до этого этапа, мы получим координаты `x` и `y`, которые являются относительными, если рассматривать их в связи со всем документом (иначе говоря, всей странице HTML, рассматриваемой как единое целое). На данный момент это не особо полезно. Нам нужны координаты относительно элемента `<canvas>` (листинг 4.26).

Листинг 4.26. Определение координат x и y

```
    x -= gCanvasElement.offsetLeft;
    y -= gCanvasElement.offsetTop;
```

Теперь у нас есть координаты x и y, привязанные к элементу <canvas>. Иначе говоря, если на текущий момент x = 0 и y = 0, то мы знаем, что пользователь только что выполнил щелчок мышью по верхнему левому пикселу холста.

Отсюда мы можем вычислить, по какому полю доски Halma пользователь щелкнул мышью, и действовать соответственно (листинг 4.27).

Листинг 4.27. Вычисление поля доски, по которому выполнен щелчок мышью

```
    var cell = new Cell(Math.floor(y/kPieceHeight),
                        Math.floor(x/kPieceWidth));
    return cell;
}
```

Ого! Оказывается, обработка событий мыши — непростая задача. Но вы можете использовать ту же самую логику (на самом деле — тот же самый код) во всех ваших приложениях, которые вы будете создавать на базе элемента <canvas>. Запомните: щелчок мышью → Координаты относительно документа → Координаты относительно холста → Код, специфичный для вашего приложения.

Хорошо, теперь давайте рассмотрим основную процедуру рисования. Поскольку графика приложения очень проста, я предпочел очищать и перерисовывать доску всякий раз, когда в игре что-то меняется:

```
gDrawingContext.clearRect(0, 0, kPixelWidth, kPixelHeight);
```

В принципе, жесткой необходимости в этом нет. Контекст рисования холста сохранит все, что вы вычертили на нем ранее, даже если пользователь прокрутит страницу так, что холст пропадет из виду или перейдет на другую вкладку, а впоследствии вернется обратно. Если вы разрабатываете приложение на базе элемента <canvas>, использующее более сложную графику (например, аркадную игру), вы можете оптимизировать производительность, отслеживая регионы холста, где происходят изменения, и перерисовывая только их. Но эта тема выходит за рамки вопросов, обсуждаемых в данной книге.

Процедура вычерчивания доски нам уже знакома. Она аналогична той, которую мы применили для вычерчивания координатной диаграммы "холста" ранее в этой главе. Эта процедура приведена в листинге 4.28.

Листинг 4.28. Процедура вычерчивания игровой доски игры Halma

```
gDrawingContext.beginPath();

/* вертикальные линии */
for (var x = 0; x <= kPixelWidth; x += kPieceWidth) {
    gDrawingContext.moveTo(0.5 + x, 0);
    gDrawingContext.lineTo(0.5 + x, kPixelHeight);
}

/* горизонтальные линии */
```

```
for (var y = 0; y <= kPixelHeight; y += kPieceHeight) {
    gDrawingContext.moveTo(0, 0.5 + y);
    gDrawingContext.lineTo(kPixelWidth, 0.5 +  y);
}

/* собственно вычерчивание */
gDrawingContext.strokeStyle = "#ccc";
gDrawingContext.stroke();
```

По-настоящему интересной ситуация становится, когда мы приступаем к рисованию отдельных фишек. Фишка представляет собой круг — кругов мы еще не вычерчивали. Более того, если пользователь выберет круг, желая выполнить ход, его надо выполнить в виде круга с заливкой цветом. Здесь аргумент `p` представляет фишку (piece), которая обладает свойствами `row` и `column`, которые обозначают текущее положение фишки на игровой доске. Мы используем некоторые внутренние константы игры для трансляции `(column, row)` в координаты "холста" `(x, y)`, после чего мы вычертим круг, а затем (если фишка выбрана) зальем этот круг сплошным цветом (листинг 4.29).

Листинг 4.29. Функция `drawPiece()`, которая вычерчивает фишку

```
function drawPiece(p, selected) {
    var column = p.column;
    var row = p.row;
    var x = (column * kPieceWidth) + (kPieceWidth/2);
    var y = (row * kPieceHeight) + (kPieceHeight/2);
    var radius = (kPieceWidth/2) - (kPieceWidth/10);
```

Это — все, что имеет отношение к логике игры. Теперь мы имеем координаты `(x, y)` центра вычерчиваемого круга, описывающие его положение на "холсте". В canvas API нет метода `circle()`, но имеется метод `arc()`. В действительности же, что представляет собой окружность, как не замкнутую дугу? Давайте вспомним базовый курс геометрии. Метод `arc()` принимает координаты центра окружности `(x, y)`, ее радиус, а также начальный и конечный углы (в радианах) плюс флаг направления (`false` в случае вращения по часовой стрелке и `true` — для вращения против часовой стрелки). Для вычисления углов, выраженных в радианах, можно применить модуль `Math`, встроенный в JavaScript (листинг 4.30).

Листинг 4.30. Вычисление углов, выраженных в радианах, с помощью встроенного модуля javaScript `Math`

```
gDrawingContext.beginPath();
gDrawingContext.arc(x, y, radius, 0, Math.PI * 2, false);
gDrawingContext.closePath();
```

Но постойте! Пока что мы еще ничего не начертили. Как и `moveTo()` и `lineTo()`, метод `arc()` представляет собой "карандашный" метод. Чтобы действительно вычертить круг, нам нужно установить свойство `strokeStyle` и вызвать метод `stroke()`, который и выполнит "обводку тушью". Это делается так, как показано в листинге 4.31.

Листинг 4.31. Фактическое вычерчивание круга

```
gDrawingContext.strokeStyle = "#000";
gDrawingContext.stroke();
```

Что делать, если фишка выделена? Мы можем повторно использовать ту же траекторию, которую мы создали для вычерчивания контура фишки, и выполнить заливку сплошным цветом (листинг 4.32).

Листинг 4.32. Повторное использование уже вычерченной траектории и заливка сплошным цветом

```
if (selected) {
    gDrawingContext.fillStyle = "#000";
    gDrawingContext.fill();
}
```

И это, в общем, все, что требуется сделать. Вся остальная часть программы реализует логику игры — определение правильных и неразрешенных ходов, отслеживание количества ходов, выявление признака завершения игры. Имея 9 кругов, несколько прямых линий и один обработчик `onclick`, мы создали настоящую игру с использованием элемента `<canvas>`. Ура!

Дополнительное чтение

- "Руководство пользователя" по работе с элементом `<canvas>` на портале разработчиков Mozilla — **https://developer.mozilla.org/en/Canvas_tutorial**
- Статья Михая Шукана (Mihai Sucan) "HTML5 canvas — the basics" — **http://dev.opera.com/articles/view/html-5-canvas-the-basics/**
- Демонстрационные примеры, инструментарий и руководства для элемента HTML5 `<canvas>` — **http://www.canvasdemos.com/**
- Описание из руководства разработчика Internet Explorer 9 — **http://msdn.microsoft.com/en-us/ie/ff468705.aspx#_HTML5_canvas**

Глава 5

Видео в Web

Любой, кто за последние четыре года посещал сайт YouTube.com, знает о том, что в Web-страницы можно встраивать видеоролики. Но до начала работы над стандартом HTML5 стандартного метода достижения этой цели определено не было. Практически все видеоролики, которые вы когда-либо просматривали в Web, "пропускались" через какой-нибудь подключаемый модуль (плагин) от стороннего разработчика — будь то QuickTime, RealPlayer или Flash (YouTube использует Flash.) Эти подключаемые модули интегрируются с вашим браузером настолько хорошо, что вы можете даже не подозревать о том, что пользуетесь ими. По крайней мере, вы можете не подозревать об этом до тех пор, пока не попытаетесь просмотреть видео на платформе, которая не поддерживает этого плагина.

HTML5 определяет стандартный способ встраивания видео в Web-страницу с помощью элемента `<video>`. Поддержка элемента `<video>` все еще находится в разработке, и это — просто "политкорректный" метод сказать, что пока что она еще не работает полноценно. По крайней мере, если она и работает, то отнюдь не везде. Но не поддавайтесь отчаянию! Существуют альтернативы, обходные методы, да и вообще — к вашим услугам самый широкий выбор разнообразных опций.

В табл. 5.1 приведено краткое описание поддержки элемента `<video>` в стандарте HTML5 большинством современных популярных браузеров и мобильных устройств.

Таблица 5.1. *Поддержка элемента <video>*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
9.0+	3.5+	3.0+	3.0+	10.5+	1.0+	2.0+

Но поддержка элемента `<video>` самого по себе — это еще не вся "история", а лишь ее часть. Прежде чем мы сможем поговорить о видео HTML5, вам необходимо разобраться с некоторыми основными понятиями, которые касаются видео как такового. Если все эти понятия вам уже знакомы, то вы можете пропустить первые вводные разделы данной главы и начинать чтение с *разд. "Что работает в Web"*.

Видеоконтейнеры

Когда речь заходит о файлах видео, вы обычно имеете в виду AVI-файлы или MP4-файлы. В действительности же AVI и MP4 — это просто контейнерные форматы. Как ZIP-архив может содержать в своем составе разнотипные файлы, так и видеофайлы в одном из контейнерных форматов могут содержать внутри себя разнотипные видеофайлы. При этом контейнерные форматы видеофайлов могут определять лишь то, *как* хранится информация, которую они содержат, но не то, *какие именно* видеоданные там содержатся. На самом деле это лишь упрощенная формулировка, а в действительности ситуация гораздо сложнее, потому что не все видеопотоки совместимы с контейнерными форматами. Но пока, чтобы не затруднять восприятие, мы не будем останавливаться на этом подробно.

Видеофайл обычно содержит множество *дорожек* — видеотреки (без аудиоинформации), плюс одну или несколько звуковых дорожек (саундтреков), не содержащих видеоинформации. Обычно эти дорожки взаимосвязаны. Звуковые дорожки содержат маркеры, упрощающие синхронизацию видео и звука. Отдельные дорожки могут содержать метаданные (metadata), такие, как соотношение сторон (пропорции) кадра видеодорожки, или же информацию о языке аудиодорожки. Контейнеры тоже могут содержать метаданные, например, заголовок самого видео, сведения об обложке видеоальбома, номера эпизодов (для телевизионных шоу) и так далее.

Существует *множество* контейнерных форматов видео. К числу наиболее популярных относятся:

- MPEG 4 (**http://en.wikipedia.org/wiki/MPEG-4_Part_14**), обычно такие файлы имеют расширение `.mp4` или `.m4v`. Контейнер MPEG 4 основан на более раннем контейнерном формате Apple, QuickTime (`.mov`). Видеоролики, скачиваемые с сайта Apple, по-прежнему могут использовать старый контейнерный формат QuickTime, но видеофайлы, которые вы получаете через iTunes, предоставляются в контейнерном формате MPEG 4.
- Flash Video (**http://en.wikipedia.org/wiki/Flash_Video**), такие файлы обычно имеют расширение `.flv`. Файлы Flash Video, что неудивительно, используются Adobe Flash. До выхода Flash 9.0.60.184 (или Flash Player 9 Update 3) это был единственный контейнерный формат, поддерживаемый Flash. Более новые версии Flash поддерживают и контейнер MPEG 4.
- Ogg (**http://en.wikipedia.org/wiki/Ogg**), обычно такие файлы имеют расширение `.ogv`. Ogg представляет собой открытый стандарт, дружественный по отношению к открытому и свободному программному обеспечению, свободный от патентных обременений. Firefox 3.5, Chrome 4 и Opera 10.5 обеспечивают встроенную поддержку контейнера Ogg, без использования зависящих от платфомы подключаемых модулей, видеофайлы Ogg (известные под названием "Theora") и аудиофайлы Ogg (известные как "Vorbis"). Для приложений, рассчитанных на настольные компьютеры, Ogg поддерживается в большинстве ведущих дистрибутивов Linux, а на платформах Mac и Windows его можно использовать, установив компоненты QuickTime (**http://www.xiph.org/quicktime/**) или фильтры

DirectShow (**http://www.xiph.org/dshow/**), соответственно. Кроме того, такие файлы могут воспроизводиться отличным мультимедийным проигрывателем VLC (**http://www.videolan.org/vlc/**), доступным для всех платформ.

- WebM (**http://www.webmproject.org/**) — новый контейнерный формат. Технически он подобен еще одному известному контейнерному формату, называющемуся Matroska (**http://en.wikipedia.org/wiki/Matroska**). Формат WebM был анонсирован в мае 2010 года. Он разработан так, чтобы использоваться исключительно с видеокодеком VP8 (**http://en.wikipedia.org/wiki/VP8**) и аудиокодеком Vorbis (чуть позже мы поговорим о кодеках более подробно). На новейших версиях браузеров Chromium, Google Chrome, Mozilla Firefox и Opera обеспечивается встроенная поддержка этого контейнерного формата, без использования плагинов, зависимых от платформы. Корпорация Adobe тоже анонсировала поддержку видео в формате WebM в будущей версии Flash.
- Audio Video Interleave (**http://en.wikipedia.org/wiki/Audio_Video_Interleave**), такие файлы обычно имеют расширение `.avi`. Контейнерный формат AVI был изобретен Microsoft еще в те времена, когда воспроизведение видео на ПК считалось экзотикой. Он не поддерживает официально таких функций, присущих современным контейнерным форматам, как, например, встроенные метаданные. Он не поддерживает на официальном уровне даже большинства из распространенных на сегодняшний день аудио- и видеокодеков. В течение некоторого времени ряд компаний предпринимал попытки расширить его таким образом, чтобы обеспечить новые кодеки, но способы эти в большинстве своем не были совместимы между собой. Этот контейнерный формат все еще используется по умолчанию такими популярными кодировщиками, как MEncoder (**http://www.mplayerhq.hu/DOCS/HTML/en/encoding-guide.html**).

Видеокодеки

Когда мы говорим о "просмотре видео", большинство людей обычно имеют в виду комбинацию из одного видеопотока и одного аудиопотока. Но у вас нет двух разных файлов, обычно у вас имеется один "видеофайл". Это может быть файл AVI или файл MP4. Это — просто контейнерные форматы, как ZIP-файл, в составе которого содержатся разнотипные файлы. Контейнерный формат определяет, каким образом видео- и аудиопотоки хранятся в составе единого файла.

Когда вы просматриваете видео, ваш видеопроигрыватель одновременно выполняет как минимум три действия:

1. Интерпретацию контейнерного формата. Это делается для обнаружения доступных видео- и аудиодорожек и определения способа их хранения в файле с тем, чтобы выяснить, нуждаются ли данные в последующем декодировании.
2. Декодирование видеопотока и отображение на экране серии изображений.
3. Декодирование аудиопотока и отправка звука на ваши динамики.

Видеокодек (video codec) — это алгоритм, с помощью которого кодируется видеопоток, иными словами, он описывает способ выполнения операции номер 2 из только что приведенного списка. (Слово "кодек" представляет собой комбинацию

из двух слов: "кодировщик" (coder) и "декодировщик" (decoder). Ваш видеопроигрыватель декодирует видеопоток в соответствии с алгоритмом, заданным вашим видеокодеком, а затем отображает на экране серию изображений — кадров или фреймов (frames). Большинство современных кодеков используют различные механизмы, позволяющие минимизировать объем информации, необходимой, чтобы отобразить кадр, следующий за предыдущим. Например, вместо сохранения каждого отдельного кадра (например, экранного снимка), они сохраняют лишь различия между кадрами. Большинство видеофайлов не содержат большого количества изменений, сильно отличающих один кадр от другого, что позволяет применять высокие степени сжатия информации. Благодаря этому результирующие файлы имеют меньшие размеры.

Различают видеокодеки с потерями информации (lossy) и без потерь (lossless). Видеокодеки, кодирующие информацию без потерь, создают видеофайлы, размер которых слишком велик, чтобы их можно было использовать в Web, так что мы в основном сосредоточим внимание на кодеках с потерями. Название "кодек с потерями" означает, что при кодировании часть информации теряется безвозвратно. Как и при копировании аудиокассет, вы теряете информацию об исходном видео и ухудшаете качество при каждой операции кодирования. Как многократно переписанная с кассет аудиозапись начинает издавать посторонние шумы, шипение и свист, так и многократно перекодированная видеозапись начинает "дергаться", особенно при воспроизведении динамических сцен. На самом деле это может произойти, даже если вы выполняете кодирование оригинальной записи, если вы выберете неподходящий кодек или неправильно зададите параметры. Положительной стороной кодеков с потерями является возможность обеспечения высокой степени сжатия за счет сглаживания "блочности" при воспроизведении, так, что потери перестают восприниматься человеческим глазом.

Существует великое множество видеокодеков (**http://samples.mplayerhq.hu/V-codecs/**). К числу трех наиболее распространенных относятся H.264, Theora и VP8, и именно их мы и рассмотрим в книге.

H.264

H.264 (**https://secure.wikimedia.org/wikipedia/en/wiki/H.264**) — это кодек, известный также под названием "MPEG-4 part 10", или "MPEG-4 AVC" (MPEG-4 Advanced Video Coding). Кодек H.264 был разработан группой MPEG (**https://secure.wikimedia.org/wikipedia/en/wiki/Moving_Picture_Experts_Group**) и стандартизован в 2003 году. Целью разработчиков являлось предоставление пользователям единого кодека для устройств с узкой полосой пропускания и ограниченными ресурсами CPU (сотовых телефонов); устройств с широкой полосой пропускания и изобилием вычислительных мощностей (современных настольных компьютеров); а также для всех промежуточных классов устройств. Чтобы добиться этой цели, разработчики стандарта H.264 разбили его на несколько "профилей" (**https://secure.wikimedia.org/wikipedia/en/wiki/H.264#Profiles**), каждый из которых определял оптимальный набор функций для файла конкретных размеров. Более "высокие" профили используют оптимизированные наборы функциональных

возможностей по выбору, предлагают лучшее качество изображения при уменьшенных размерах файлов, требуют больше времени на кодирование и потребляют больше ресурсов процессора при декодировании в режиме реального времени.

Чтобы составить приблизительное представление о диапазоне профиля, скажем, что Apple iPhone (**http://www.apple.com/iphone/specs.html**) поддерживает базовый профиль (Baseline profile), приставка AppleTV set-top box (**http://www.apple.com/appletv/specs.html**) поддерживает профили базовый (Baseline) и основной (Main), а Adobe Flash (**http://www.kaourantin.net/2007/08/what-just-happened-to-video-on-web_20.html**) на настольном ПК поддерживает базовый и основной профили, а также "высокий" (High) профиль. YouTube сейчас применяет H.264 для кодирования видео высокого разрешения (high-definition videos, HD)[1], воспроизводимого с помощью Adobe Flash. Кроме того, YouTube предоставляет видео, закодированное с помощью H.264, для воспроизведения на мобильных устройствах, включая Apple iPhone и смартфоны, работающие под управлением мобильной операционной системы Google Android (**https://code.google.com/android/**). Кроме того, H.264 является одним из видеокодеков, соответствующих спецификации Blu-Ray (**http://en.wikipedia.org/wiki/Blu-ray_Disc**); диски Blu-Ray, использующие этот кодек, обычно предназначены для устройств, поддерживающих профиль High.

Большинство устройств, отличных от ПК, воспроизводящих видео H.264 (включая iPhone и независимые проигрыватели Blu-Ray), выполняют декодирование с помощью встроенного аппаратного чипа, поскольку их процессоры не обладают вычислительными мощностями, требующимися для декодирования видео в режиме реального времени. В наши дни даже бюджетные графические карты поддерживают аппаратное декодирование H.264. Существует множество конкурирующих кодировщиков H.264 (**http://compression.ru/video/codec_comparison/mpeg-4_avc_h264_2007_en.html**), включая библиотеку x264 (**http://www.videolan.org/developers/x264.html**) на основе открытого кода. Стандарт H.264 имеет патентные обременения; вопросы лицензирования решает группа MPEG LA (**http://www.mpegla.com/main/default.aspx**). Видеофайлы H.264 могут встраиваться в наиболее популярные контейнерные форматы, включая MP4, используемый в онлайновом магазине Apple's iTunes (**http://www.apple.com/itunes/whatson/movies.html**), и MKV (в основном применяемый энтузиастами некоммерческого видео).

Theora

Theora (**https://secure.wikimedia.org/wikipedia/en/wiki/Theora**) ведет происхождение от кодека VP3 (**https://secure.wikimedia.org/wikipedia/en/wiki/Theora#History**). Этот формат был разработан некоммерческой организацией Xiph.org Foundation (**http://xiph.org/**). Theora представляет собой бесплатный кодек, не обремененный никакими из известных патентных ограничений, кроме оригинальных патентов VP3, которые лицензируются бесплатно. Хотя с 2004 года стандарт был "заморожен", проект Theora (включая эталонные кодировщик и декодер на основе откры-

[1] См. **http://www.wired.com/epicenter/2008/12/youtube-adds-hd/**.

того кода) выпустил версию 1.0 в ноябре 2008 (**http://lists.xiph.org/pipermail/theora-dev/2008-November/003736.html**), а версию 1.1 — в сентябре 2009 (**http://lists.xiph.org/pipermail/theora-dev/2009-September/003985.html**).

Видео Theora может встраиваться в любой контейнерный формат, хотя чаще всего встречается в контейнерах Ogg. Все основные дистрибутивы Linux обладают встроенной поддержкой Theora, а браузер Mozilla Firefox 3.5 включает поддержку видео Theora (**http://lists.xiph.org/pipermail/theora-dev/2009-September/003985.html**) в контейнере Ogg. Под "встроенной поддержкой" я понимаю немедленную доступность на всех платформах без необходимости использования сторонних плагинов. Видео Theora можно воспроизводить в Windows (**http://www.xiph.org/dshow/**) или Mac OS X (**http://xiph.org/quicktime/**), если установить в этих системах декодер Xiph.org на основе открытого свободно распространяемого кода.

VP8

VP8 (**https://secure.wikimedia.org/wikipedia/en/wiki/VP8**) — это еще один видеокодек разработки On2, той же компании, которая ранее создала кодек VP3 (который впоследствии получил известность под именем Theora). С технической точки зрения он производит видео такого же уровня, как и H.264 High Profile, но при этом сложность преобразований сравнима с H.264 Baseline.

В 2010 году компания Google приобрела On2 и опубликовала спецификацию кодека, а также перевела кодировщик и декодер в разряд открытого, свободно распространяемого ПО. В порядке реализации этой инициативы, компания Google открыла и патенты, полученные On2 на кодек VP8, начав их бесплатное лицензирование. Это — лучшее из того, на что вы можете надеяться, имея дело с патентами. Вы не можете их ликвидировать или "аннулировать", поскольку они выпущены и присвоены. Чтобы сделать их дружественными по отношению к открытому и свободному ПО, вы бесплатно лицензируете их, и после этого все стороны, заинтересованные в использовании технологий, "покрываемых" патентами, смогут использовать их бесплатно, без необходимости платить лицензионные сборы и отчисления. По состоянию на 19 мая 2010 года, VP8 представляет собой современный кодек, свободный от лицензионных отчислений, и не обременен никакими другими ограничениями, кроме патентов, выданных компании On2 (на настоящий момент приобретенной Google), которые лицензируются, как уже говорилось, бесплатно.

Аудиокодеки

Если только вы не собираетесь ограничиться просмотром фильмов, снятых примерно до 1927 года (**http://www.filmsite.org/jazz.html**), то вам, в дополнение к видеодорожке, понадобится еще и саундтрек. Как и видеокодеки, аудиокодеки представляют собой алгоритмы, в соответствии с которыми кодируется аудиопоток. Как и видеокодеки, аудиокодеки подразделяются на кодеки, выполняющие кодирование с потерями (lossy) и кодеки без потерь (lossless). Как и видеокодеки без потерь, аудиокодеки без потерь создают файлы, размер которых слишком велик

для того, чтобы их можно было использовать в Web. Поэтому основное внимание мы сосредоточим на кодеках, выполняющих кодирование с потерями.

На самом деле мы еще более сузим диапазон рассматриваемых кодеков, потому что аудиокодеки с потерями подразделяются на различные категории. Аудиопотоки используются и там, где о видео вопрос даже не ставится (например, в телефонии), и существует целая категория кодеков, оптимизированных для кодирования речи (**http://www.voip-info.org/wiki/view/Codecs**). С помощью таких кодеков ни один человек в здравом уме не станет выполнять, например, захват (риппинг) с музыкального CD, потому что результат окажется еще хуже по качеству, чем пение четырехлетнего ребенка в телефонную трубку. Но они найдут себе применение, например, в Asterisk PBX[1] (**http://www.asterisk.org/**), потому что полоса пропускания — ценный ресурс, а эти кодеки могут сжимать человеческую речь до файлов таких размеров, которые составляют десятые доли от размеров файлов, получаемых при помощи кодеков общего назначения. Однако вследствие недостаточной встроенной поддержки как в браузерах, так и в сторонних плагинах, кодеки, оптимизированные для кодирования человеческой речи, практически никогда не встречаются в Web. Поэтому я буду рассматривать, в основном, распространенные аудиокодеки общего назначения, выполняющие кодирование с потерями.

Как я уже говорил ранее, когда вы смотрите видео, ваш компьютер одновременно выполняет три задачи, как минимум:

1. Интерпретирует контейнерный формат.
2. Декодирует видеопоток.
3. Декодирует аудиопоток и отправляет звук на воспроизводящие устройства.

Аудиокодек описывает выполнение шага 3 — декодирование аудиопотока, преобразование его в цифровой аудиосигнал, который на звуковоспроизводящем устройстве преобразуется в звук. Как и в случае с видеокодеками, все эти операции производятся, чтобы уменьшить объем информации в аудиопотоке. Поскольку мы говорим об аудиокодеках с потерями, во время цикла "Запись → Кодирование → Декодирование → Прослушивание" происходят потери информации. Различные кодеки по-разному обрабатывают информацию и отбрасывают разные ее фрагменты, но все они преследуют одну и ту же цель: "обмануть" уши слушателей таким образом, чтобы они не догадались о том, что некоторые фрагменты звуковой информации отсутствуют.

Кроме того, в области аудио есть еще одна концепция, которая отсутствует в обработке видео — *каналы* (channels). Мы отправляем звук на динамики, разве не так? Хорошо, а сколько у вас динамиков? У человека, сидящего за настольным компьютером, их, как правило, два: левый и правый. Но у меня, например, их даже три — один слева, второй — справа, а третий — сзади на полу. Так называемые

[1] Asterisk IP-PBX — свободное решение компьютерной телефонии с открытым исходным кодом от компании Digium, первоначально разрабатываемое Марком Спенсером (Mark Spencer). Приложение работает в Linux, FreeBSD и Solaris. Asterisk, в комплексе с необходимым оборудованием, обладает всеми возможностями классической АТС, поддерживает множество протоколов VoIP и предоставляет функции голосовой почты, конференций, интерактивного голосового меню и прочие функции. Подробнее см. **https://secure.wikimedia.org/wikipedia/en/wiki/Asterisk_%28PBX%29**. — *Прим. перев.*

системы "объемного звука" (surround sound)[1] могут иметь шесть или даже большее количество колонок-динамиков, с большим знанием дела расположенных в пределах помещения. На каждый динамик подается звук с определенного канала оригинальной записи. С точки зрения теории, вы можете сидеть в центре участка, окруженного шестью динамиками, каждый из которых воспроизводит свой канал. Фактически, вы окружены шестью звуковоспроизводящими устройствами, а особенности восприятия звука человеческим мозгом создают иллюзию "присутствия" там, где происходит действие. Это работает? В принципе, то, что в бизнесе, занимающемся системами объемного звучания, крутятся миллионы долларов, говорит нам о том, что да, эта концепция "работает".

Большинство многоцелевых аудиокодеков массового потребления рассчитаны на работу с двумя звуковыми каналами. Во время записи звук разбивается на два канала; в процессе кодирования оба канала сохраняются в едином звуковом потоке; в процессе декодирования каналы декодируются по отдельности, и каждый из них направляется на соответствующий динамик. Некоторые аудиокодеки могут иметь более двух каналов, при этом они отслеживают, на какой динамик следует направлять конкретный канал, так, что ваш проигрыватель имеет возможность правильно распределять каналы по динамикам.

Как уже говорилось, аудиокодеков существует великое множество (**http://wiki.multimedia.cx/index.php?title=Category:Audio_Codecs**). Я говорил вам о том, что видеокодеков тоже великое множество? Так забудьте об этом. Что касается аудиокодеков, то в Web вы чаще всего будете сталкиваться с тремя из них: MP3, AAC и Vorbis.

MPEG-1 Audio Layer 3

MPEG-1 Audio Layer 3 (**https://secure.wikimedia.org/wikipedia/en/wiki/MPEG-1_Audio_Layer_3**) — это кодек, повсеместно известный под более "разговорным" названием MP3. Если вы до сих пор еще ничего не слышали про MP3, то лично я вообще не знаю, чему я смогу вас научить. В крупных магазинах электроники продается множество разнообразных портативных проигрывателей, которые известны под собирательным названием "MP3-плееры".

MP3-файлы могут содержать до 2 звуковых каналов. Они могут кодироваться с различными битрейтами: 64 kbps, 128 kbps, 192 kbps, а также на множестве других скоростей из диапазона от 32 до 320. Чем выше скорость битового потока, тем больше размер результирующего файла и тем выше качество звучания, хотя соотношение между скоростью битового потока (битрейтом) и качеством звука носит нелинейный характер. Например, качество файла, закодированного на 128 kbps, почти вдвое выше, чем для файла, закодированного на 64 kbps, но для битрейтов 256 kbps и 128 kbps это различие уже менее заметно. Более того, формат MP3 позволяет выполнять кодирование с переменным битрейтом, когда некоторые части закодированного потока сжимаются сильнее других. Например, паузы могут кодироваться на более низком битрейте, а когда множество инструментов начинают ис-

[1] См. **https://secure.wikimedia.org/wikipedia/en/wiki/Surround_sound**.

полнять сложную композицию, битрейт резко взлетает. MP3-файлы могут кодироваться и с постоянным битрейтом (constant bitrate encoding).

Стандарт MP3 не дает абсолютно точного определения метода кодирования файлов, хотя для декодирования файлов такое определение дается. Различные кодировщики используют разные психоакустические модели, неудивительно поэтому то, что они дают и разные результаты, но все получаемые файлы декодируются одними и теми же проигрывателями. Лучшим среди бесплатных кодировщиков является созданный в рамках проекта LAME (**http://lame.sourceforge.net/**) на основе открытого свободного кода. Он же, очевидно, является и лучшим среди всех кодировщиков, если не считать кодирование на низких битрейтах.

Формат MP3 (стандартизованный в 1991 году) обременен патентами, что и служит причиной того, что Linux не может воспроизводить MP3-файлы сразу же после установки (так как нет встроенной поддержки). Практически любой портативный музыкальный проигрыватель поддерживает независимые MP3-файлы, и аудиопотоки MP3 могут встраиваться в любой видеоконтейнер. Adobe Flash может воспроизводить как независимые MP3-файлы, так и аудиопотоки в составе видеоконтейнера MP4.

Advanced Audio Coding

Формат Advanced Audio Coding (**https://secure.wikimedia.org/wikipedia/en/wiki/Advanced_Audio_Coding**) широко известен под названием AAC. Стандартизованный в 1997 году, он стал вездесущим после того, как компания Apple выбрала его в качестве формата по умолчанию для своего сервиса iTunes Store. Изначально все AAC-файлы, приобретенные через iTunes Store, были зашифрованы с помощью патентованой DRM-схемы Apple, известной под названием FairPlay (**https://secure.wikimedia.org/wikipedia/en/wiki/FairPlay**). Сейчас избранные музыкальные номера, предлагающиеся в iTunes Store, являются бесплатными и поставляются как незащищенные AAC-файлы, которые Apple называет "iTunes Plus" (и в самом деле, не назовете же вы бонусную программу "iTunes Minus"). Формат AAC обременен патентами; расценки на лицензирование доступны в режиме online (**http://www.vialicensing.com/licensing/aac-fees.aspx**).

Кодек AAC разрабатывался с тем, чтобы обеспечить более высокое качество звучания, чем MP3 при одинаковых битрейтах, и он действительно позволяет кодировать звуковую информацию на любом битрейте. Для сравнения, формат MP3 ограничен фиксированным количеством битрейтов, причем верхняя граница допустимого диапазона составляет 320 kbps. AAC позволяет кодировать до 48 звуковых каналов включительно, хотя на практике этого не делает никто. Формат AAC отличается от формата MP3 еще и тем, что он определяет множество *профилей* (profiles), во многом аналогично H.264, и по тем же самым причинам. Профиль "пониженной сложности" (low-complexity) предназначается для воспроизведения файлов в режиме реального времени на устройствах с пониженной мощностью CPU, тогда как более "высокие" профили предлагают улучшенное качество звучания при тех же битрейтах. Платить за это приходится пониженными скоростями кодирования и декодирования.

На данный момент все современные продукты компании Apple, включая iPod, AppleTV и QuickTime, поддерживают определенные профили AAC как в независимых аудиофайлах, так и для аудиопотоков в составе видеоконтейнера MP4. Adobe Flash поддерживает все профили AAC в MP4, как и видеопроигрыватели на основе открытого кода MPlayer и VLC. Для кодирования может применяться открытая библиотека FAAC (**http://sourceforge.net/projects/faac/**); ее поддержка может быть включена в `mencoder` (**https://secure.wikimedia.org/wikipedia/en/wiki/MEncoder**) и `ffmpeg` (**https://secure.wikimedia.org/wikipedia/en/wiki/FFmpeg**) во время компиляции из исходного кода.

Vorbis

Vorbis (**https://secure.wikimedia.org/wikipedia/en/wiki/Vorbis**) часто называют "Ogg Vorbis", хотя с технической точки зрения это некорректно, потому что Ogg представляет собой только контейнерный формат, и аудиопотоки Vorbis могут встраиваться не только в этот, но и в другие контейнеры. Vorbis не обременен никакими из известных патентов, и поэтому все ведущие дистрибутивы Linux, как и все известные портативные устройства, использующие свободную прошивку Rockbox (**http://www.rockbox.org/**), на основе открытого кода обеспечивают его встроенную поддержку. Mozilla Firefox 3.5 поддерживает аудиофайлы Vorbis в контейнере Ogg или видеофайлы Ogg с аудиодорожкой Vorbis. Мобильные телефоны Android (**https://code.google.com/android/**) тоже могут воспроизводить независимые аудиофайлы Vorbis. Аудиопотоки Vorbis обычно встраиваются в контейнеры Ogg или WebM, но могут встраиваться и в контейнеры MP4 (**http://samples.mplayerhq.hu/MPEG-4/vorbis-in-mp4/**), в MKV, или даже, с некоторыми хакерскими трюками — в AVI (**http://www.alexander-noe.com/video/amg/**). Vorbis поддерживает произвольное количество звуковых каналов.

Существуют открытые кодировщики и декодеры Vorbis, в том числе такие, как кодировщик `OggConvert` (**http://oggconvert.tristanb.net/**), декодер `ffmpeg` (**http://www.ffmpeg.org/**), кодировщик `aoTuV` (**http://www.geocities.jp/aoyoume/aotuv/**) и декодер `libvorbis` (**http://downloads.xiph.org/releases/vorbis/**). Кроме того, существуют компоненты QuickTime для Mac OS X (**http://www.xiph.org/quicktime/**) и фильтры DirectShow для Windows (**http://www.xiph.org/dshow/**).

Что работает в Web

Если у вас до сих пор еще не зарябило в глазах от всего этого многообразия, значит, ваша выносливость к восприятию информации явно выше среднего. Как видите, видео и аудио — это сложный вопрос, причем в данной книге я привел весьма упрощенный обзор данной темы! Теперь у вас должен возникнуть законный вопрос — а как все это соотносится с HTML5. В HTML5 имеется элемент `<video>`, позволяющий встраивать в вашу страницу видеоролики. На используемые в элементе `<video>` видеокодеки, аудиокодеки и контейнерные форматы не налагается

никаких ограничений. Один элемент может содержать ссылки на множество видеофайлов, и браузер выберет первый видеофайл, который он фактически сможет воспроизвести. Обязанность определять, какие браузеры обеспечивают поддержку конкретных контейнеров и кодеков, возлагается на вас.

На момент написания этих строк картина с поддержкой браузерами видео HTML5 была такой:

- Mozilla Firefox (3.5 и более новые версии) поддерживает видео Theora и аудио Vorbis в контейнере Ogg. Firefox 4 поддерживает и WebM.
- Opera (10.5 и более поздние версии) поддерживает видео Theora и аудио Vorbis в контейнере Ogg. Opera 10.60 поддерживает и WebM.
- Google Chrome (3.0 и более поздние версии) поддерживает видео Theora и аудио Vorbis в контейнере Ogg. Кроме того, поддерживаются видео H.264 (все профили) и аудио AAC (все профили) в контейнере MP4. Google Chrome 6.0 также поддерживает WebM.
- Safari на компьютерах Mac и ПК, работающих под управлением Windows (версия 3.0 и более новые), будет поддерживать все, что поддерживается QuickTime. Теоретически вы можете потребовать, чтобы ваши пользователи установили сторонние плагины QuickTime. но имейте в виду, что на практике это сделают лишь немногие. По этой причине можете считать, что вы остались "один на один" с теми форматами, которые обеспечивают встроенную поддержку QuickTime. Список таких форматов довольно длинен, но в него не входят такие контейнеры, как WebM, Theora, Vorbis или Ogg. Тем не менее, QuickTime поддерживает видео H.264 (основной профиль) и аудио в формате AAC в составе контейнера MP4.
- Мобильные телефоны наподобие Apple iPhone и смартфоны Google Android поддерживают видео H.264 (базовый профиль) и аудио AAC ("облегченный" профиль) в контейнере MP4.
- Adobe Flash (9.0.60.184 и более поздние версии) поддерживает видео H.264 (все профили) и аудио AAC (все профили) в контейнере MP4.
- Internet Explorer 9 будет поддерживать некоторые из еще не документированных профилей видео H.264 и аудио AAC в контейнере MP4.
- Internet Explorer 8 вообще не поддерживает видео по стандарту HTML5, но практически все пользователи Internet Explorer устанавливают плагин Adobe Flash. Далее в этой главе я продемонстрирую, как вы сможете пользоваться видео HTML5, корректно прибегая к Flash при необходимости.

Возможно, что вся эта информация будет удобнее для восприятия, если ее перевести в табличную форму. Краткая сводка информации о поддержке видео HTML5 современными браузерами и мобильными устройствами приведена в табл. 5.2.

Уже через год с настоящего момента эта картина радикальным образом изменится, так как поддержка WebM будет реализована во многих браузерах, и они будут предоставлять стандартизованные, а не экспериментальные версии WebM, а пользователи постепенно начнут переход на новые версии браузеров.

Краткая сводка информации о планируемой поддержке видеокодеков в предстоящих релизах браузеров приведена в табл. 5.3.

Таблица 5.2. *Поддержка видеокодеков в современных браузерах*

Кодеки/контейнер	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora+Vorbis+Ogg	·	3.5+	†	5.0+	10.5+	·	·
H.264+AAC+MP4	·	·	3.0+	5.0‡	·	3.0+	2.0+
WebM	·	·	†	6.0+	10.6+	·	·

† Safari может воспроизводить все, что может воспроизводиться с помощью QuickTime. QuickTime поставляется с встроенной поддержкой H.264/AAC/MP4. Существуют устанавливаемые сторонние плагины, которые добавляют поддержку для Theora и WebM, но для того, чтобы браузер Safari начал распознавать эти видеоформаты, пользователь должен был установить эти плагины.

‡ Google Chrome вскоре прекратит поддержку H.264 (см. **http://blog.chromium.org/2011/01/html-video-codec-support-in-chrome.html**). Более подробное объяснение причин можно найти здесь: **http://blog.chromium.org/2011/01/more-about-chrome-html-video-codec.html**.

Таблица 5.3. *Поддержка видеокодеков в предстоящих версиях браузеров*

Кодеки/контейнер	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora+Vorbis+Ogg	·	3.5+	†	5.0+	10.5+	·	·
H.264+AAC+MP4	9.0+	·	3.0+	5.0+	·	3.0+	2.0+
WebM	9.0+*	4.0+	†	6.0+	10.6+	·	2.3‡

* Internet Explorer 9 будет поддерживать WebM, только "если пользователь установил кодек VP8", а это подразумевает, что Microsoft не будет поставлять этот кодек самостоятельно.

† Safari будет воспроизводить все, что может быть воспроизведено с помощью QuickTime, но QuickTime обеспечивает встроенную поддержку только для H.264/AAC/MP4.

‡ Хотя Android 2.3 уже поддерживает WebM, аппаратных декодеров пока нет, поэтому срок работы от батарей представляет большую проблему.

А теперь подведем итоги:

> ***Замечание Профессора Разметки***
>
> Единой комбинации кодеков и контейнеров, которая работала бы во всех браузерах HTML5, не существует.
>
> В ближайшем будущем это положение дел вряд ли изменится.
>
> Чтобы ваше видео можно было просматривать на всех видах устройств и на всех платформах, вам придется кодировать ваши видеофайлы несколько раз.

Для обеспечения максимальной совместимости рекомендуется придерживаться следующего порядка при обработке видео:

1. Создайте одну версию, использующую WebM (VP8 + Vorbis).
2. Создайте еще одну версию, использующую базовый профиль видео H.264 и "упрощенный" профиль аудио AAC в контейнере MP4.
3. Создайте еще одну версию, использующую видео Theora и аудио Vorbis в контейнере Ogg.
4. Дайте ссылки на все три видеофайла из единого элемента `<video>`, и прибегните как к резерву к видеопроигрывателю на базе Flash.

Вопросы лицензирования для видео H.264

Прежде чем переходить к дальнейшему изложению, я должен заметить, что двойное кодирование ваших видео связано с издержками. Хотя очевидных затрат и нет, но уже само то, что вы должны несколько раз перекодировать свои видеофайлы, означает, что вам требуется больше времени и больше компьютеров, а ведь это — тоже затраты. Помимо прочего, с видео H.264 сопряжены и реальные затраты — на лицензирование.

Когда я впервые упомянул видео H.264, я сказал и о том, что этот видеокодек обременен патентами и лицензированием управляет консорциум MPEG LA. Это очень важно. Чтобы понять, почему это важно, пройдитесь по лабиринту лицензирования H.264[1]: MPEG LA разбивает весь пакет лицензий на H.264 на две подлицензии: одну — для производителей кодировщиков и декодеров, и еще одну — для тех, кто распространяет контент.

> "Подлицензия для распространителей контента, в свою очередь, разбивается на 4 подкатегории, две из которых (распространение по подписке и покупка отдельных номеров, или платное пользование) связаны с тем, платит ли конечный пользователь непосредственно за услуги видеосервиса, а еще две ("бесплатное" телевидение и интернет-трансляции) привязаны к иным источникам, а не к конечному потребителю.
>
> […]
>
> Лицензионные отчисления за "бесплатное" телевидение базируются на двух опциях выплаты. Первая из них представляет собой единоразовую выплату в сумме $2,500 на пропускание видеопотока через кодировщик передачи AVC, что покрывает расходы "потребителя лицензии на одноразовую передачу AVC-видео конечному пользователю", который сможет декодировать и просмотреть видео. Если вас интересует вопрос о том, не является ли это двойным взиманием платы за одну и ту же услугу, то ответ будет положительным. Да, это так и есть: лицензионные отчисления уже получены от производителя кодировщика, а компания, осуществляющая трансляцию, выплатит лицензионные отчисления еще раз: в соответствии с одним из выбранных вариантов.
>
> Второй лицензионной выплатой является плата за годовую лицензию на трансляцию. […] Годовая лицензия на трансляцию разбивается на подопции в соответствии с размером аудитории:
>
> - $2,500 за календарный год на рынке, насчитывающем от 100,000 до 499,999 телеприемников;
> - $5,000 за календарный год для вещательного рынка, насчитывающего от 500,000 до 999,999 телеприемников;
> - $10,000 за календарный год для вещательной компании, чья аудитория насчитывает от 1,000,000 зрителей.
>
> […] Имея в виду все эти тонкости, касающиеся "бесплатного" телевидения, зададимся вопросом: а какое отношение ко всему этому имеют те, кто не зани-

[1] См. **http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/The-H.264-Licensing-Labyrinth-65403.aspx**.

мается вещанием? Как я уже упоминал ранее, плата "за участие" относится ко всем, кто в той или иной форме производит и поставляет контент. После того как консорциум MPEG LA определил, что "бесплатное" телевидение означает не просто "эфирное вещание", он пошел дальше и определил отчисления "за участие" для тех, кто занимается трансляциями через Интернет в форме "трансляции видео AVC для конечных пользователей по всему миру через Интернет, для случаев, когда конечный пользователь не платит за право приема и просмотра". Иными словами, любая форма широковещательной рассылки, включая эфирное вещание, кабельные каналы, спутниковые каналы или Интернет, требует выплаты лицензионных отчислений. […]

При этом расценки за распространение через Интернет потенциально выше, возможно, потому, что объем интернет-трансляций растет быстрее, чем объем трансляций OTA (Over-the-Air) или "бесплатных" телетрансляций по сетям кабельного телевидения либо спутниковым каналам. Если сложить отчисления рынка "бесплатного телевидения" и лицензионные отчисления за вещание через Интернет, MPEG LA дает гарантию, что "после истечения первого лицензионного периода (который закончился 31 декабря 2010 года), "лицензионные отчисления не превысят экономический эквивалент роялти, выплачиваемых за тот же период за бесплатное телевидение".

Последняя часть, касающаяся лицензионных отчислений за интернет-вещание, уже добавлена. Недавно консорциум MPEG-LA объявил, что за потоковую передачу через Интернет плата взиматься не будет. Но это не означает, что кодек H.264 свободен от лицензионной платы для всех пользователей. В частности, кодировщики (подобные используемому для закачивания видео на сервис YouTube) и декодеры (подобные включенному в браузер Google Chrome) облагаются лицензионными выплатами. Более подробную информацию можно найти в заметке "Бесплатно, как дымовая завеса" (Free as in smokescreen), вот здесь: **http://shaver.off.net/diary/2010/08/27/free-as-in-smokescreen/**.

Кодирование видео с помощью конвертера Miro

Существует множество инструментов для кодирования видео, и на качество кодирования влияют многие опции. Если вы не ставите перед собой цели разобраться во всех деталях кодирования видео, а просто хотите, как потребитель, кодировать некоторые свои ролики, то вам адресован именно данный раздел.

Miro Video Converter (**http://www.mirovideoconverter.com/**) — это программа на основе открытого исходного кода, поставляемая по лицензии GPL, предназначенная для кодирования видео во множестве различных форматов. Для скачивания доступны версии для Windows и Mac OS X. Программа поддерживает абсолютно все форматы вывода, упомянутые в данной главе. Никаких опций, кроме выбора видеофайла и выводного формата, она не предлагает. В качестве ввода программа принимает абсолютно любой видеофайл, включая видео в формате DV (**http://en.wikipedia.org/wiki/DV**), создаваемое большинством камкордеров потребительского уровня. В большинстве слу-

чаев программа обеспечивает вывод приемлемого качества. Впрочем, из-за того, что программа не предоставляет практически никаких опций, если выходное качество вас не удовлетворяет, вам не останется ничего другого, кроме как перейти на использование другой программы.

Чтобы начать работу, запустите приложение Miro Video Converter. На экране появится главное окно приложения (рис. 5.1).

Щелкните по ссылке **Choose a file** и выберите видеофайл, который вам требуется закодировать (рис. 5.2).

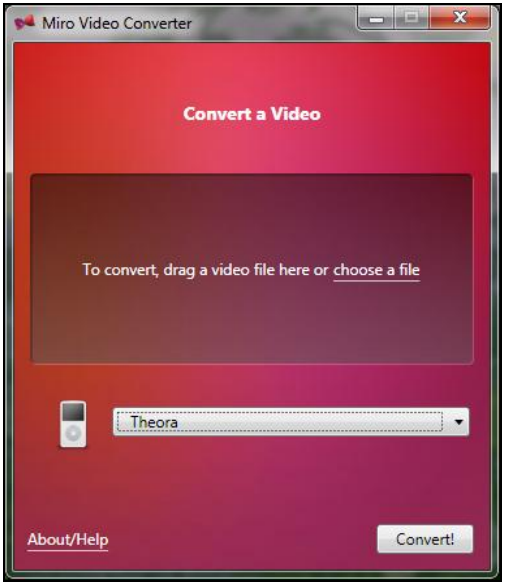


Рис. 5.1. Главное окно приложения Miro Video Converter

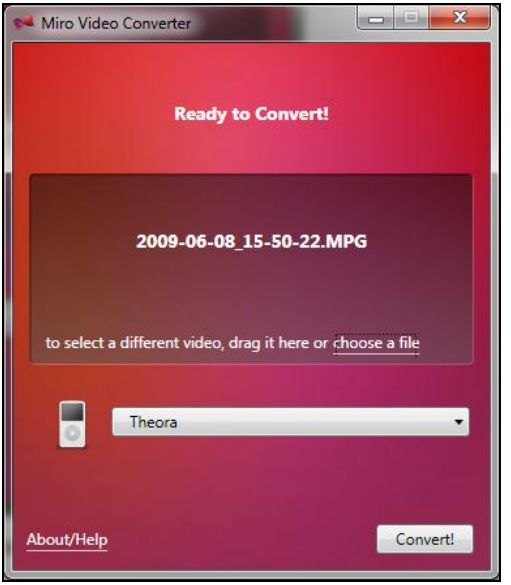


Рис. 5.2. Выбор файла для кодирования

Расположенный ниже раскрывающийся список содержит список выводных форматов и устройств, для которых можно выполнить кодирование (рис. 5.3).

Для целей нашей главы мы заинтересованы только в трех из них:

- **WebM (vp8)** — эта опция соответствует кодированию в формат WebM: видеокодек VP8 и аудио Vorbis в контейнере WebM.
- **Theora** — эта опция соответствует видео Theora и аудио Vorbis в контейнере Ogg.
- **iPhone** — эта опция соответствует базовому профилю видео H.264 и облегченному профилю AAC в контейнере MP4.

Сначала выберите опцию WebM (см. рис. 5.3), после чего нажмите кнопку **Convert!**, и программа немедленно начнет процедуру кодирования вашего видео. Выводной файл будет сохранен в том же каталоге, что и исходный видеофайл, под именем *SOURCEFILE*`.webmvp8.webm`, где *SOURCEFILE* — имя исходного видеофайла.

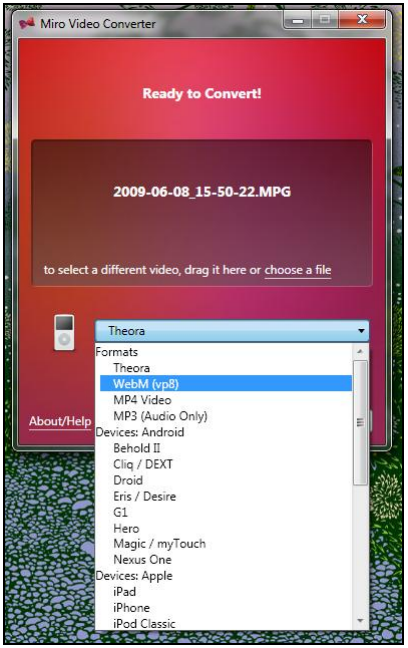


Рис. 5.3. Выбор выводных форматов и устройств

Рис. 5.4. Miro Video Converter ведет кодирование видеофайла — созерцать этот экран вам придется довольно долго

Рис. 5.5. На этот раз выберите опцию **Theora**

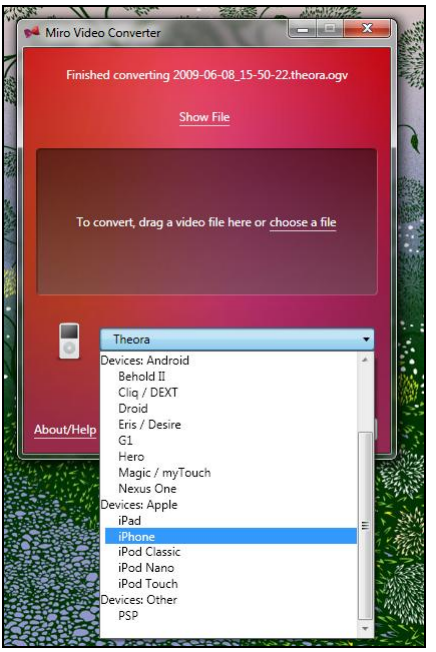


Рис. 5.6. Выбор опции **iPhone**

Как только кодирование завершится, вы вернетесь в главное окно. Снова выберите файл для кодирования, но на этот раз выберите из раскрывающегося списка устройств и форматов опцию **Theora** (рис. 5.5).

Снова нажмите кнопку **Convert!**, и Miro Video Converter выполнит кодирование вашего видеофайла в формат Theora. Выходной файл получит имя *SOURCEFILE*`.theora.ogv`, где *SOURCEFILE* — имя исходного видеофайла. Пока идет преобразование, вы вполне сможете успеть выпить кофе.

Наконец, когда кодирование завершится, выберите из списка устройств и форматов опцию **iPhone**, чтобы получить видео в формате H.264, совместимое с iPhone (рис. 5.6).

Для видео, совместимого с iPhone, Miro Video Converter предлагает опцию отправки закодированного файла в вашу библиотеку iTunes (рис. 5.7). Выбирать эту опцию или не выбирать — целиком зависит только от вас, но для публикации вашего видео в Web это совершенно не обязательно.

Снова нажмите магическую кнопку **Convert!** и опять подождите, пока Miro Video Converter завершит процедуру преобразования (рис. 5.8). Закодированный видеофайл будет сохранен в том же каталоге, что и исходный, и получит имя *SOURCENAME*`.iphone.mp4`, где *SOURCENAME* — это имя исходного файла.

Рис. 5.7. Для видео, совместимого с iPhone, предлагается опция отправки закодированного файла в вашу библиотеку iTunes (для публикации видео в Web в этом нет необходимости)

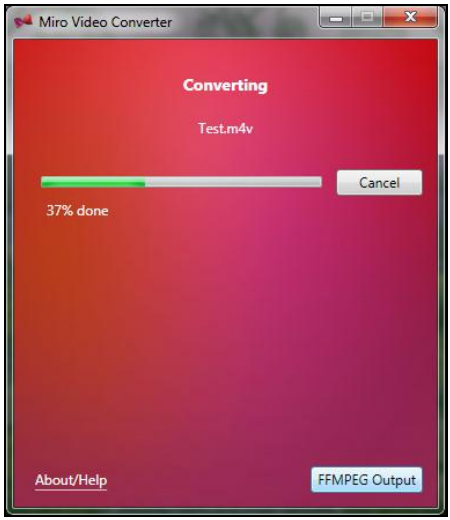


Рис. 5.8. Преобразование больших файлов может оказаться длительным процессом

Теперь кроме исходного видеофайла у вас будут еще три закодированных, пригодных для публикации в Web. Если вы вполне удовлетворены качеством видео, можете больше ни о чем не беспокоиться и продолжать чтение с раздела "Наконец, перейдем к разметке", где будет рассказано о том, как собрать все эти файлы

в составе общего элемента <video>, который будет поддерживаться всеми браузерами. Если качество видео, полученного с помощью Miro Video Converter, вас не устраивает или если вы любознательны и интересуетесь и другим, более сложным инструментарием, продолжайте читать следующие разделы.

Кодирование видео Ogg с помощью Firefogg

В данном разделе я собираюсь использовать термин "видео Ogg" как сокращенное наименование для термина "видео Theora и аудио Vorbis в контейнере Ogg". Эта комбинация кодеков и контейнера получает встроенную поддержку в таких браузерах, как Mozilla Firefox и Google Chrome.

Firefogg (**http://firefogg.org/**) представляет собой расширение Firefox на основе открытого кода, лицензирующееся по схеме GPL и предназначенное для кодирования видео Ogg. Чтобы воспользоваться этим решением, вам потребуется установить Mozilla Firefox 3.5 или более новую версию (**http://www.mozilla.com/ru/firefox/?from=getfirefox**), а затем посетить домашнюю страницу проекта **firefogg.org** (рис. 5.9).

Щелкните мышью по ссылке **Install Firefogg**. Firefox выведет запрос о том, действительно ли вы желаете установить данное расширение (рис. 5.10). Для продолжения нажмите кнопку **Разрешить** (Allow).

Рис. 5.9. Домашняя страница проекта Firefogg

Рис. 5.10. Чтобы позволить Firefox установить данное расширение, нажмите кнопку **Разрешить** (Allow)

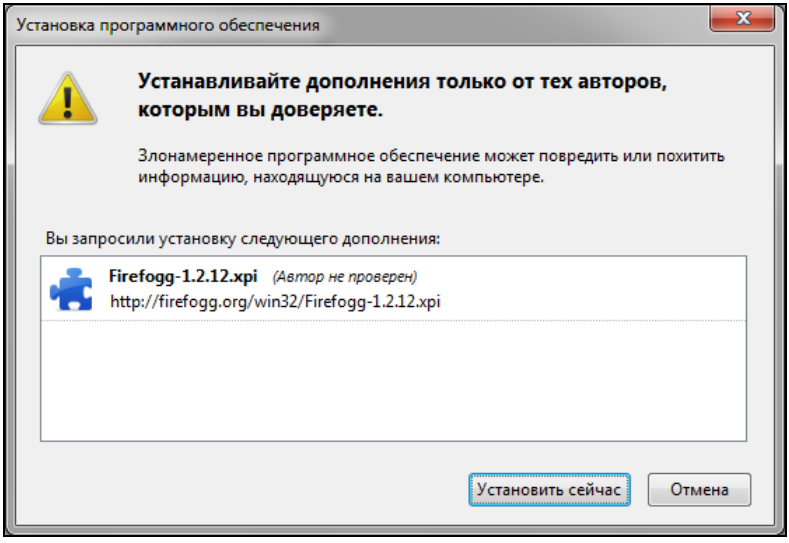


Рис. 5.11. Установка расширения Firefogg

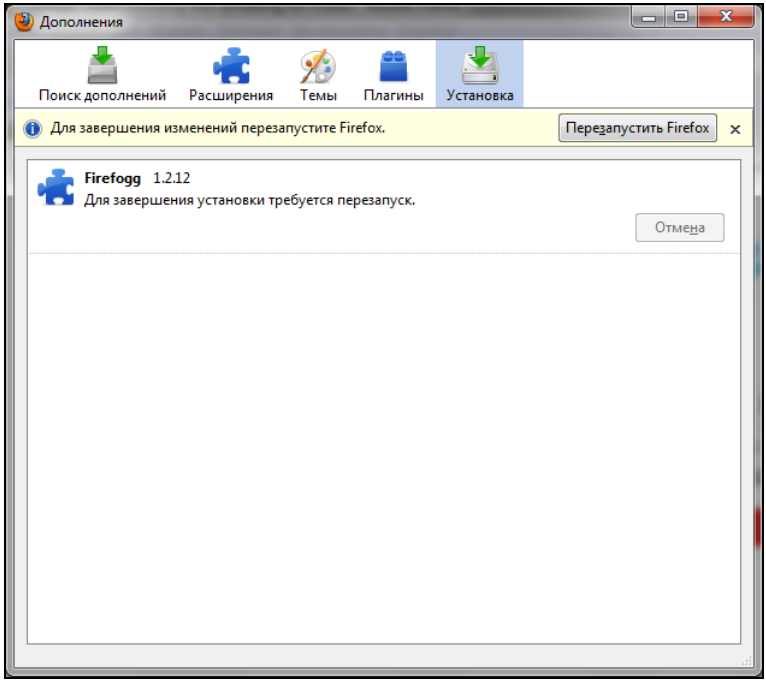


Рис. 5.12. Перезапуск Firefox

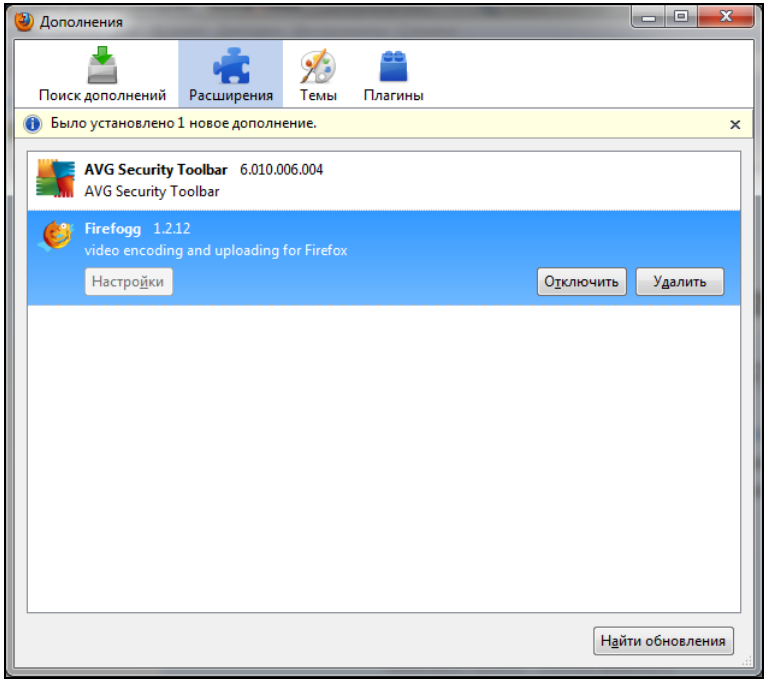


Рис. 5.13. Установка завершилась успешно

Рис. 5.14. Теперь давайте создадим видеофайл!

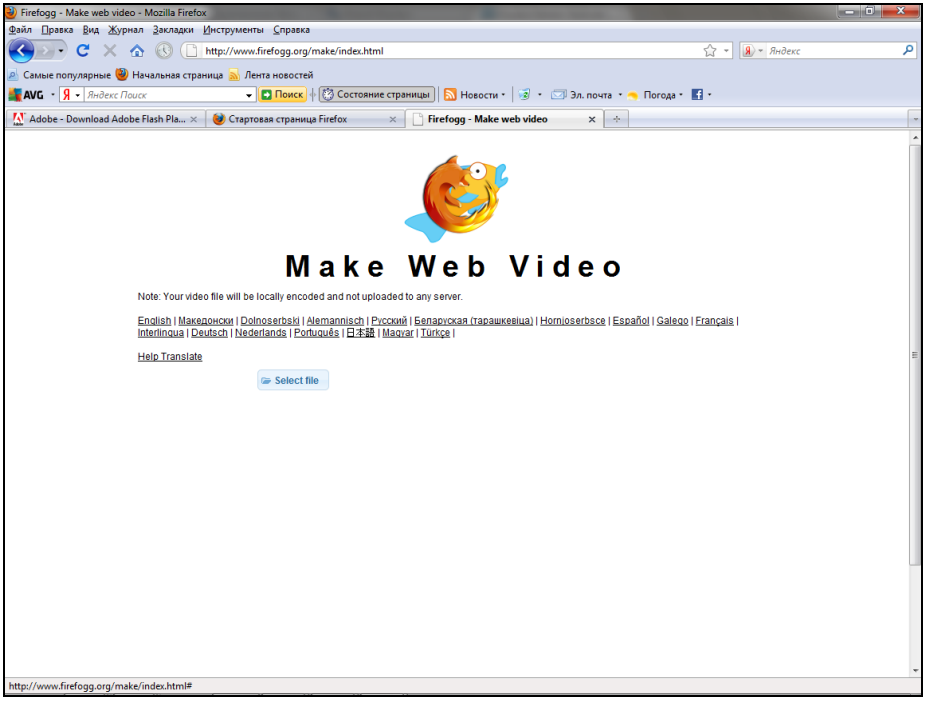


Рис. 5.15. Выбор видеофайла

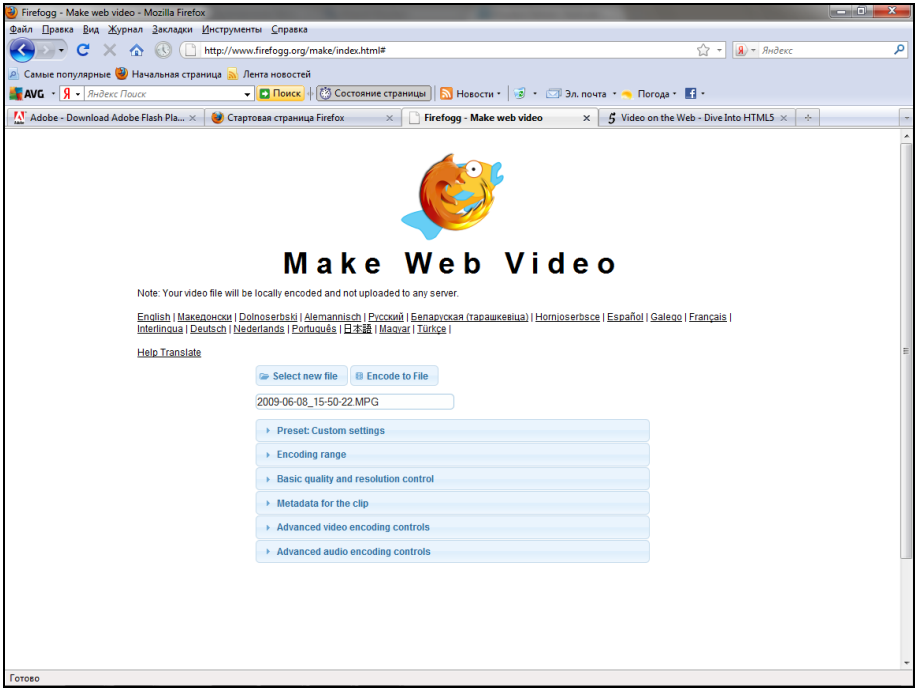


Рис. 5.16. Основные опции Firefogg для кодирования видеофайлов

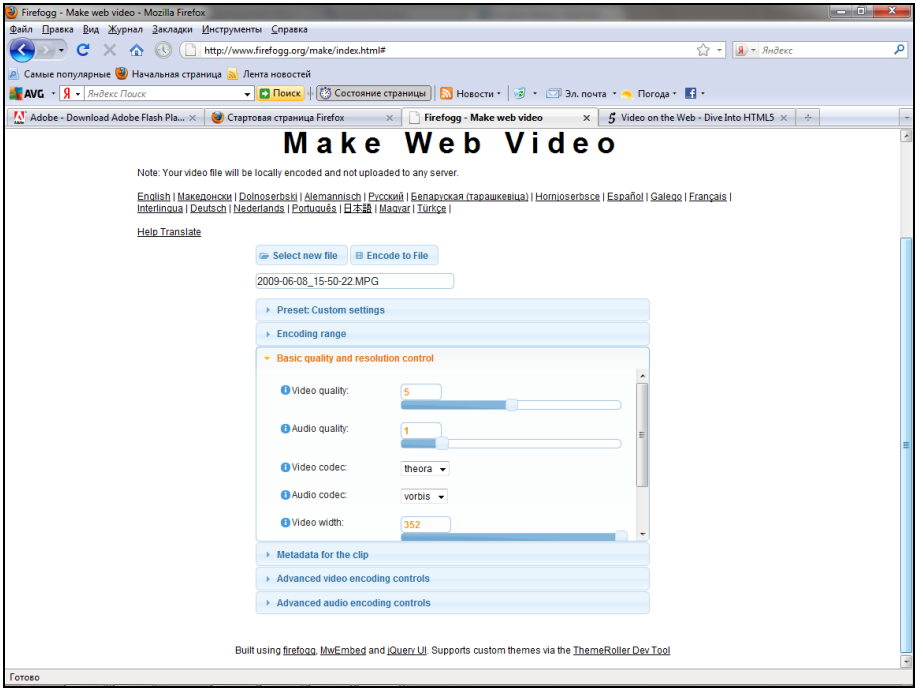


Рис. 5.17. Базовые параметры управления качеством и разрешением видео

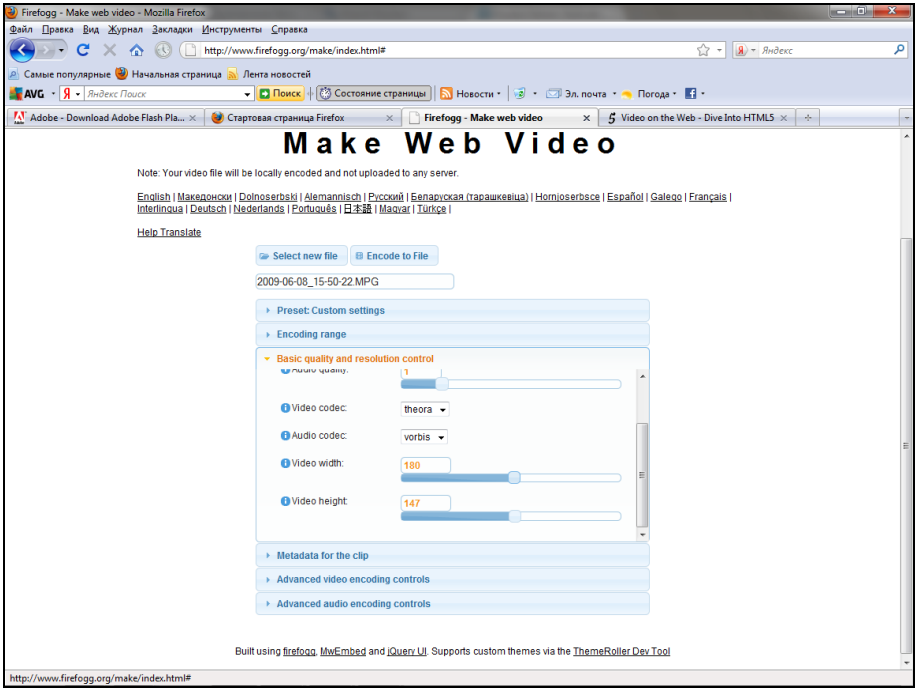


Рис. 5.18. Настройка ширины и высоты видеокадра

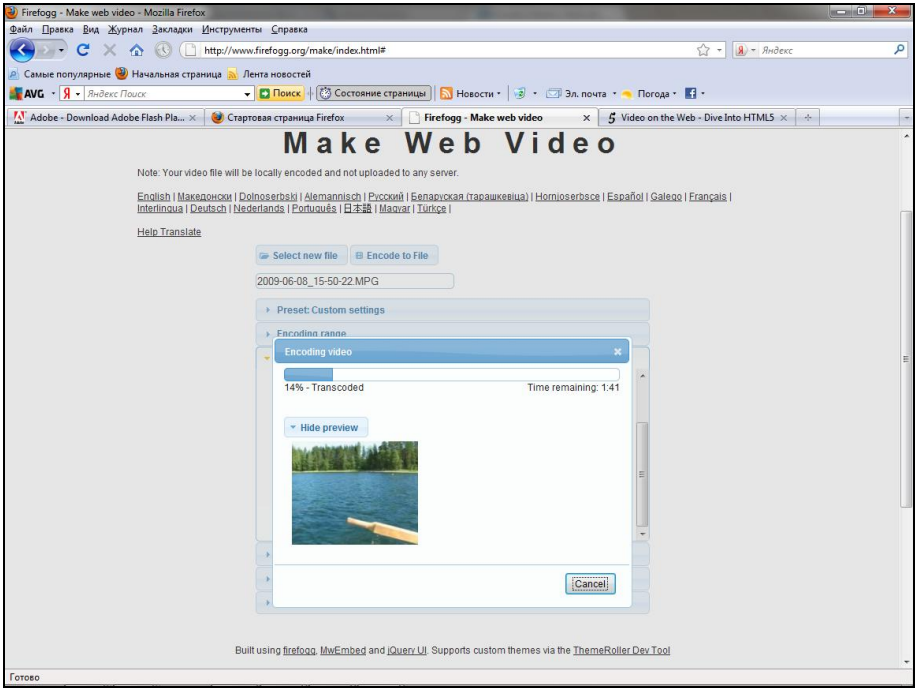


Рис. 5.19. Процесс кодирования видеофайла

После этого вы увидите стандартное окно, которое Firefox выводит, когда вы решаете установить какое-либо дополнительное программное обеспечение (рис. 5.11). Для продолжения нажмите кнопку **Установить сейчас** (Install Now).

После того как расширение будет установлено, щелкните мышью по кнопке **Перезапустить Firefox** (Restart Firefox), чтобы завершить процедуру установки (рис. 5.12).

После перезапуска Firefox, **firefogg.org** подтвердит успешность установки расширения Firefogg (рис. 5.13).

Чтобы начать процесс кодирования, щелкните по опции **Make web Video** (рис. 5.14).

Чтобы выбрать исходный файл, щелкните по кнопке **Select file** (рис. 5.15).

Firefogg имеет шесть "вкладок" (рис. 5.16):

1. **Preset: Custom settings**. Стандартный набор заранее подобранных параметров — это набор **Ogg web video Theora, Vorbis**, который вполне подходит для наших целей.
2. **Encoding range**. Кодирование видео может оказаться довольно длительным процессом. Приступая к работе, вы обычно хотите закодировать небольшой фрагмент вашего видео, чтобы ознакомиться с тем, какие результаты вы можете получить (для примера, возьмем первые 30 секунд вашего видеоролика). Возможно, вам захочется провести несколько экспериментов, чтобы выбрать оптимальную для вас комбинацию параметров.
3. **Basic quality and resolution control**. На этой вкладке, управляющей базовым уровнем качества и разрешением, находятся наиболее важные параметры.
4. **Metadata**. Здесь я не буду рассматривать данный вопрос подробно, но вы можете добавлять к своему закодированному видео различные метаданные, например, заголовки и сведения об авторах. Вполне возможно, что вы уже добавляли метаданные к своим аудиоколлекциям, собранным с помощью iTunes или любого похожего ПО для организации музыкальных коллекций. Достаточно сказать, что здесь используется та же самая идея.
5. **Advanced video encoding controls**. Не следует менять настройки на этой вкладке, за исключением тех случаев, когда вы очень хорошо понимаете, что делаете. Firefogg предлагает интерактивную справочную систему по большинству этих опций. Чтобы узнать больше о конкретной опции, щелкните по кнопке с символом **i**, расположенной справа от заинтересовавшей вас опции.
6. **Advanced audio encoding controls**. Как и в предыдущем случае, для редактирования этих опций необходимо понимать смысл и назначение каждой из них.

Я рассмотрю подробно только опции, расположенные на вкладке **Basic quality and resolution control** (рис. 5.17).

Здесь вы найдете следующие базовые настройки:

- **Video Quality**. Качество видео задается по шкале, охватывающей значения от 0 (самое низкое качество) до 10 (наивысшее качество). Более высокие значения задают большие размеры, так что вам потребуется экспериментально определить наилучшее для вас соотношение между размером и качеством.
- **Audio Quality**. Качество звука задается по шкале от –1 (низшее качество) до 10 (высшее качество). Более высокие значения означают большие размеры, как и в случае с видео.

- **Video Codec**. Для этого параметра всегда должно выбираться значение **theora**.
- **Audio Codec**. Для этого параметра всегда должно выбираться значение **vorbis**.
- **Video Width and Video Height**. Эти параметры задают фактические ширину и высоту кадра для вашего исходного видео. Если вы хотите изменить размеры кадра при кодировании, вы можете изменить значение ширины или высоты. Firefogg автоматически изменит значение другого параметра так, чтобы сохранить исходные пропорции (с тем, чтобы ваше изображение не оказалось непропорционально "сплюснутым" или "вытянутым").

В рассматриваемом примере я собираюсь изменить размеры так, чтобы ширина кадра составляла половину от исходного значения ширины. Обратите внимание на то, как Firefogg автоматически настроит высоту, чтобы сохранить оригинальные пропорции (рис. 5.18).

После настройки всех параметров нажмите кнопку **Encode to File**, чтобы начать процесс кодирования. Firefogg предложит вам указать имя результирующего файла, который будет содержать ваше закодированное видео.

Firefogg будет отображать ход процесса с помощью симпатичного индикатора прогресса (рис. 5.19). Теперь вам останется только дождаться завершения кодирования. При кодировании длинного фрагмента это может занять довольно длительное время.

Пакетное кодирование видео Ogg Video с помощью ffmpeg2theora

Как и в предыдущем разделе, здесь я буду использовать термин "Ogg video" как сокращенную форму для "Видео Theora и аудио Vorbis audio в контейнере Ogg". Эта комбинация кодеков и контейнера будет иметь встроенную поддержку в таких браузерах, как Mozilla Firefox и Google Chrome.

Если вам требуется кодировать большое количество видеофайлов Ogg, и вы хотите автоматизировать этот процесс, вам определенно следует присмотреться к программе `ffmpeg2theora`.

`ffmpeg2theora` (**http://v2v.cc/~j/ffmpeg2theora/**) — это приложение на основе открытого кода, поставляемое по лицензии GPL (**http://en.wikipedia.org/wiki/GNU_General_Public_License**) и предназначенное для Ogg. Доступны готовые сборки для Mac OS X, Windows и современных дистрибутивов Linux. Эта программа может принимать на обработку в качестве входных данных практически любой видеофайл, включая файлы в формате DV, получаемые с помощью камкордеров потребительского класса.

Чтобы начать использование `ffmpeg2theora`, вам необходимо вызвать программу из командной строки. На платформе Mac OS X это делается через меню **Applications → Utilities → Terminal**. В Windows это делается через меню **Start → Programs → Accessories → Command Prompt**.

`ffmpeg2theora` может принимать большое количество аргументов командной строки. Чтобы просмотреть подробную информацию обо всех опциях командной

строки, дайте команду `ffmpeg2theora --help`. Здесь я подробно остановлюсь лишь на самых употребительных опциях.

- `--video-quality Q`, где `Q` представляет собой число из диапазона 0 — 10.
- `--audio-quality Q`, где `Q` представляет собой число из диапазона –2 — 10.
- `--max_size=WxH`, где `W` и `H` задают максимальную ширину и высоту кадра для вашего видео. (Буква `x` между опциями — это действительно всего лишь буква `x`.) Утилита командной строки `ffmpeg2theora` меняет пропорции видеокадра таким образом, чтобы кадр попадал в указанный диапазон размеров, так что размер кадра в результирующем видеофайле может оказаться меньше, чем W×H. Например, кодирование видеофайла, имеющего размеры кадра 720×480, с помощью команды `--max_size 320×240` даст вам результирующий файл, в котором размеры кадра составят 320×213.

Таким образом, еще один способ, с помощью которого вы можете закодировать видео с такими же параметрами, как в предыдущем разделе (кодирование с помощью Firefogg), представлен в листинге 5.1.

Листинг 5.1. Кодирование видео из командной строки с помощью команды `ffmpeg2theora`

```
you@localhost$ ffmpeg2theora --videoquality 5
                             --audioquality 1
                             --max_size 320x240
                             pr6.dv
```

Закодированный видеофайл будет сохранен в том же каталоге, что и исходный, но с расширением имени файла `.ogv`. Вы можете указать другой каталог и/или имя файла, передав в командной строке `ffmpeg2theora` следующий аргумент `--output=/path/to/encoded/video`.

Кодирование видео H.264 Video с помощью HandBrake

В данном разделе я буду использовать термин "видео H.264" в качестве сокращения для термина "базовый профиль видео H.264 и облегченный профиль аудио AAC в контейнере MPEG-4". Эта комбинация кодеков и контейнера в Safari, в Adobe Flash, на устройствах iPhone и Google Android.

Оставив в стороне вопросы лицензирования, отметим, что простейший способ кодирования видео H.264 предоставляется программой HandBrake. HandBrake (**http://handbrake.fr/**) представляет собой бесплатное приложение на основе открытого кода, поставляемое по лицензии GPL и предназначенное для кодирования видео H.264. Более ранние версии могли кодировать и другие видеоформаты, но в последней версии разработчики отбросили поддержку большинства других форматов и сосредоточили все усилия на видео H.264. Готовые сборки (**http://handbrake.fr/downloads.php**) доступны для Windows, Mac OS X и современных дистрибутивов Linux.

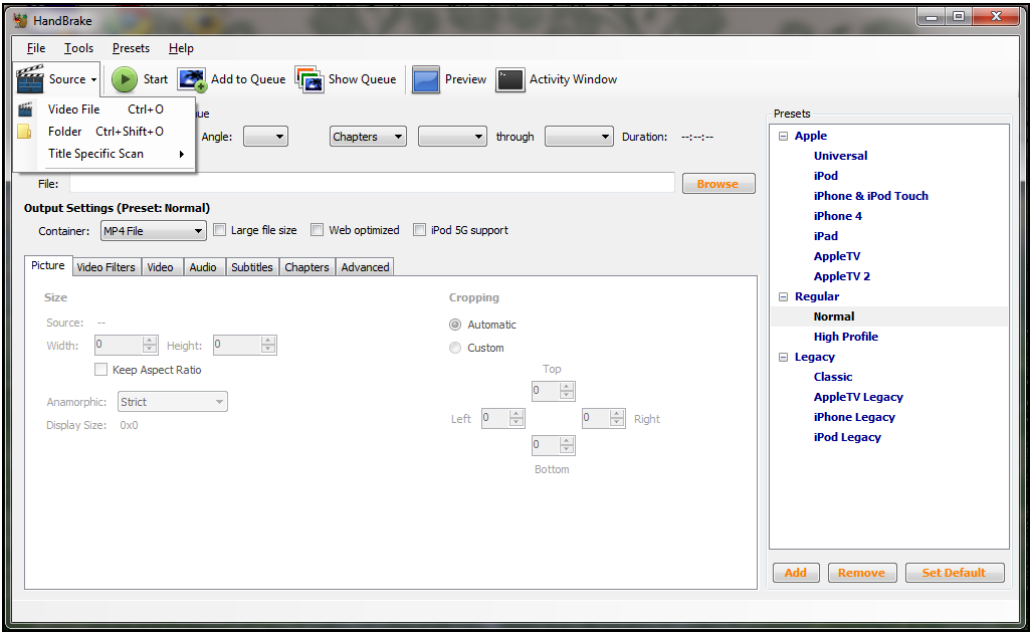


Рис. 5.20. Выберите исходный видеофайл

HandBrake поставляется в двух вариантах: с графическим интерфейсом и в виде утилиты командной строки. Сначала я опишу работу с графическим пользовательским интерфейсом, а затем мы рассмотрим, как рекомендованные настройки графической утилиты соотносятся с опциями командной строки.

Когда вы запустите приложение HandBrake, то первое, что вам необходимо будет сделать — это выбрать исходный видеофайл (рис. 5.20). Щелкните по кнопке раскрывающегося списка **Source** и выберите опцию **Video File**, чтобы выбрать исходный файл. HandBrake может принимать на обработку в качестве исходного материала практически любые видеофайлы, в том числе — видеофайлы в формате DV, получаемые с помощью камкордеров потребительского уровня.

HandBrake сообщит, что вам необходимо указать каталог по умолчанию для сохранения ваших закодированных видеофайлов (рис. 5.21). Вы спокойно можете игнорировать это сообщение или же раскрыть окно опций (через меню **Tools**) и задать каталог по умолчанию для вывода файлов.

В правой части окна находится список предопределенных наборов параметров. Если вы выберете набор предопределенных параметров **iPhone & iPod Touch**, то вы получите большинство необходимых вам опций (рис. 5.22).

Одна из важных предопределенных опций, которая по умолчанию отключена — это опция **Web optimized**. Выбор этой опции переупорядочит некоторые из метаданных в составе закодированного видеофайла так, что вы сможете просматривать начальную, уже загруженную часть видео, пока остальная продолжает загружаться в фоновом режиме. Я настоятельно рекомендую вам включить эту опцию (рис. 5.23). Она не влияет ни на качество, ни на размер закодированного видеофайла, поэтому отказываться от нее нет никакого смысла.

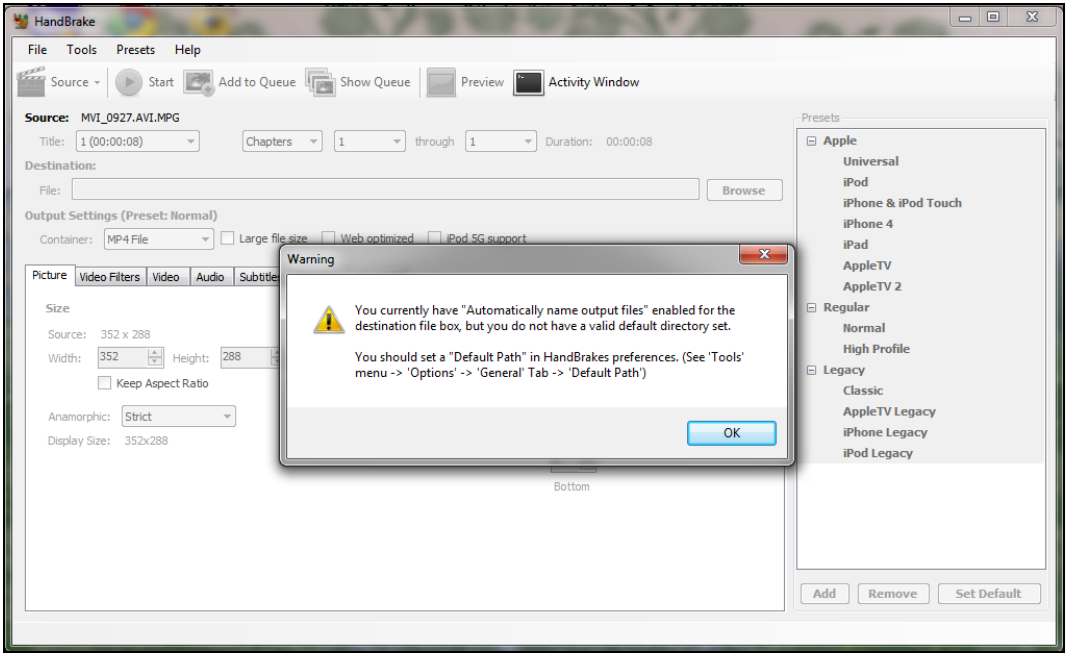


Рис. 5.21. Это сообщение можно проигнорировать

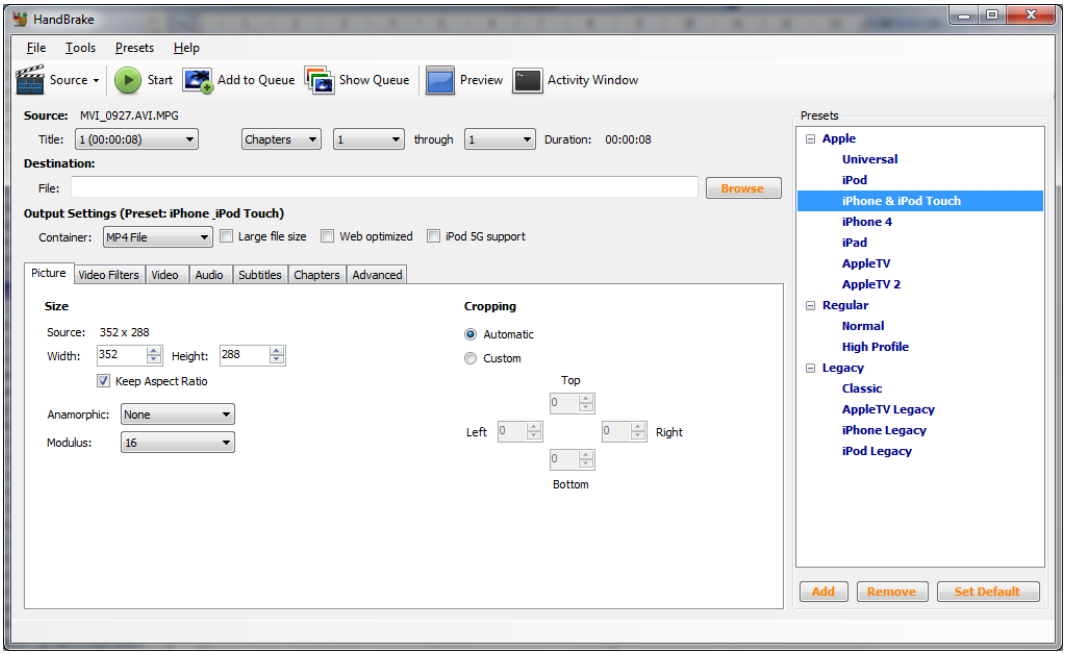


Рис. 5.22. Выбор набора предопределенных опций для iPhone

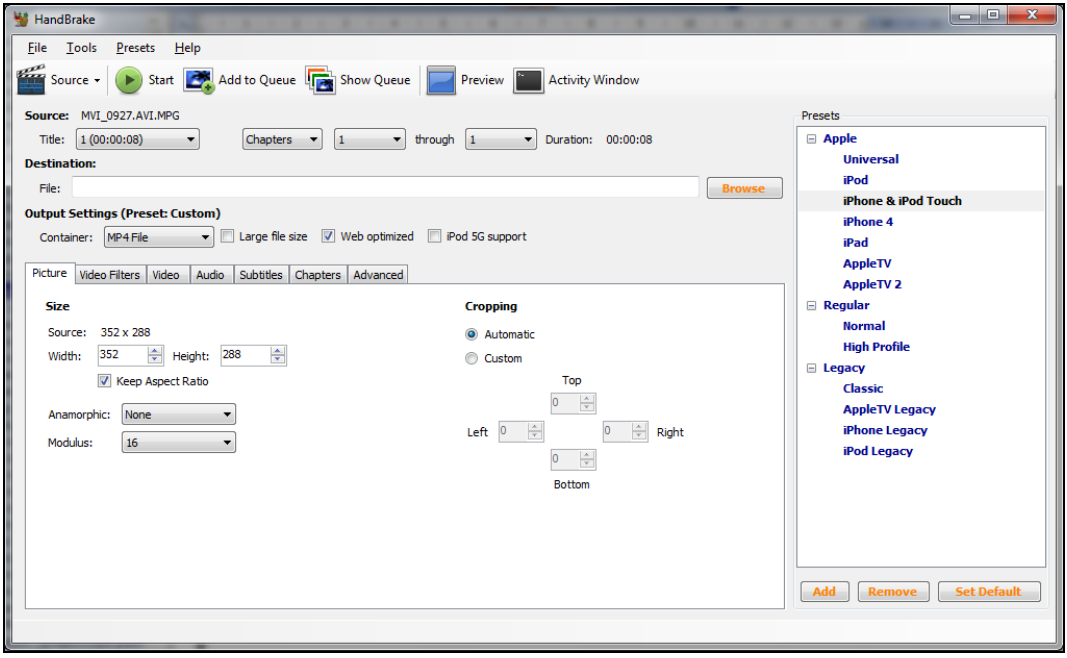


Рис. 5.23. Всегда выполняйте оптимизацию вашего файла для Web

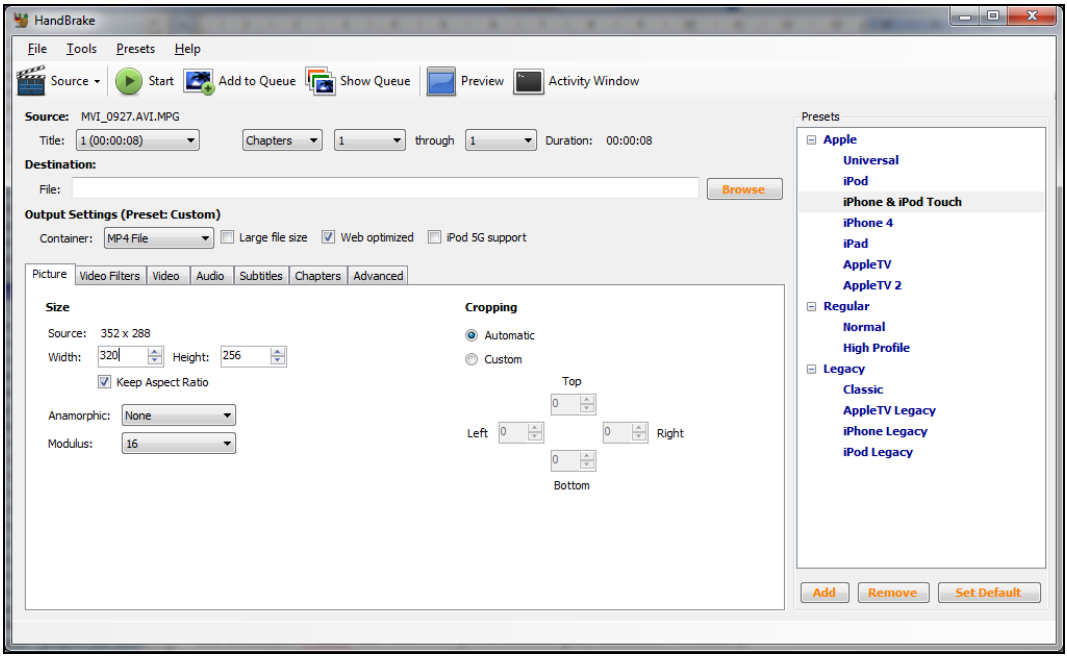


Рис. 5.24. Установка максимальных ширины и высоты кадра

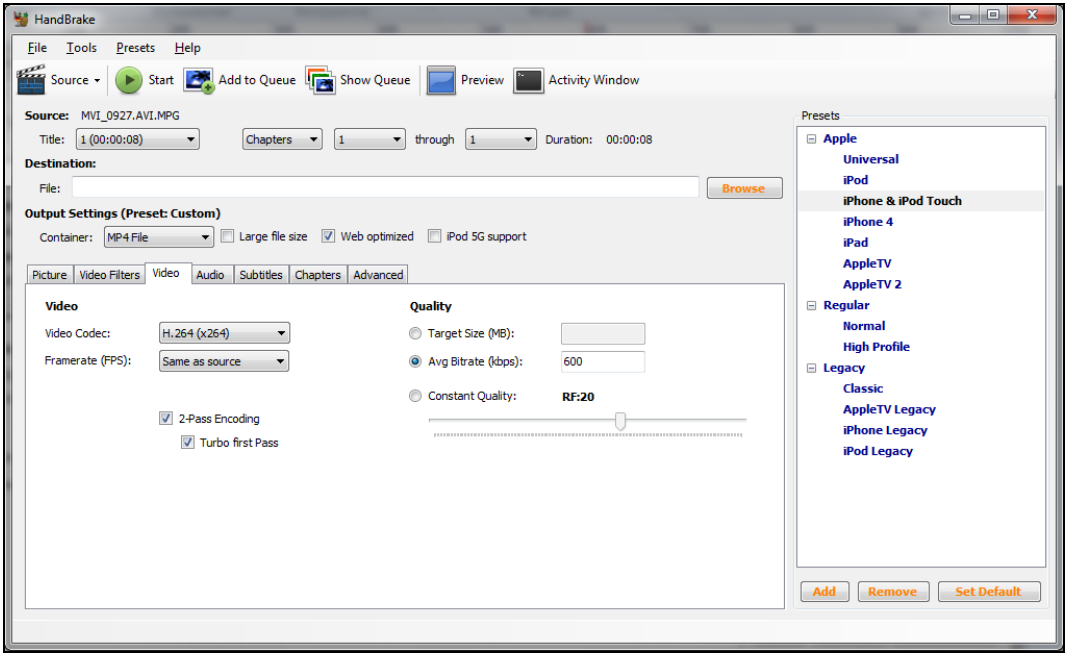


Рис. 5.25. Опции качества видео

На вкладке **Picture** вы можете задать максимальную ширину и высоту кадра закодированного видео. Я рекомендую всегда устанавливать опцию **Keep Aspect Ratio**, чтобы гарантировать, что HandBrake не исказит пропорции кадра при перемасштабировании (рис. 5.24).

На вкладке **Video** (рис. 5.25) вы можете установить ряд важных опций:

- **Video Codec**. Убедитесь, что выбрана опция H.264 (x264).
- **2-Pass Encoding**. Если этот флажок установлен, HandBrake запустит видеокодировщик дважды. При первом запуске кодировщик просто проанализирует видео в поисках такой информации, как цветовая композиция, движение (motion) и прерывания между сценами (scene breaks). На втором проходе будет осуществлено фактическое кодирование видео с использованием информации, полученной на первом проходе. Как и следовало бы ожидать, это займет примерно вдвое больше времени, чем однопроходное кодирование, зато в результате вы получите видеофайл более высокого качества без увеличения его размеров. Я всегда активизирую двухпроходное кодирование для видео H.264. Если только вы не стремитесь создать самостоятельно нечто, подобное YouTube и не колируете видео все 24 часа в сутки, возможно, опция двухпроходного кодирования подойдет и вам тоже.
- **Turbo First Pass**. После того как вы активизируете опцию двухпроходного кодирования, вы получите возможность немного ускорить обработку, выбрав опцию **Turbo first pass**. Эта опция уменьшает объем работы, выполняемой на первом проходе (анализ видео), лишь ненамного понижая качество результата.

Я обычно устанавливаю эту опцию, но если качество для вас — наивысший приоритет, тогда вам следует оставить ее блокированной.

- **Quality**. Есть различные способы указать качество для вашего закодированного видео. Вы можете указать размер целевого файла, который ваш результирующий видеофайл не должен превышать, и программа HandBrake постарается сделать все, от нее зависящее, чтобы размер результирующего файла не превзошел указанный вами. Вы можете указать средний "битрейт", который фактически означает количество бит, которое должно быть сохранено на секунду закодированного видео. Этот показатель называется "средним" битрейтом, потому что некоторые секунды закодированного потребуют большего количества бит, чем другие. Или же вы можете указать постоянное качество, по шкале от 0 до 100%. Более высокие значения соответствуют более высокому качеству, но при этом будут получаться и файлы большего размера. Единственно правильного ответа на вопрос о том, какое значение качества в каждом конкретном случае следует выбрать именно вам, не существует.

Вопрос к Профессору Разметки

В: Можно ли использовать двухпроходное кодирование и в случае с видео Ogg?

О: Да, но вследствие фундаментальных отличий в принципе работы кодировщика, вам, возможно, это и не понадобится (см. **http://en.flossmanuals.net/TheoraCookbook/FFMPEG2Theora**). Двухпроходная схема кодирования H.264 практически всегда повышает качество выходного видеофайла. Двухпроходная схема кодирования файлов видео Ogg полезна только тогда, когда вы пытаетесь добиться, чтобы выходной файл имел конкретный размер. Возможно, это именно то, в чем вы заинтересованы, но это — не то, что демонстрируется в рассматриваемых примерах, и, кроме того, возможно, что результат (видео для Web) не оправдывает дополнительного расхода времени. Чтобы добиться максимального качества видео Ogg, следует пользоваться настройками качества видео, и не тратить время на двухпроходное кодирование (см. **http://hacks.mozilla.org/2009/09/theora-1-1-released/**).

В рассматриваемом примере, я выбрал значение среднего битрейта 600 kbps, что достаточно неплохо для закодированного видео 320×240. Далее в этой главе будет продемонстрирован пример с кодированием видео на 200 kbps. Я выбрал двухпроходную схему кодирования с первым проходом в режиме "турбо".

На вкладке **Audio** (рис. 5.26) вам, возможно, не потребуется менять ничего. Если ваш исходный видеофайл имеет множество аудиодорожек, возможно, вам понадобится выбрать, какая из них должна быть включена в ваш закодированный видеофайл. Если в вашем видео основное звуковое сопровождение — это человеческая речь (в противоположность музыке или звуковому фону окружающей среды), то возможно, что вы сможете уменьшить битрейт примерно до 96 kbps. За исключением этого, все остальные настройки по умолчанию, унаследованные вами от набора предопределенных параметров **iPhone & iPod Touch**, должны вам подойти.

Далее, щелкните по кнопке **Browse** и выберите каталог для сохранения видеофайла и имя, под которым он должен быть сохранен (рис. 5.27).

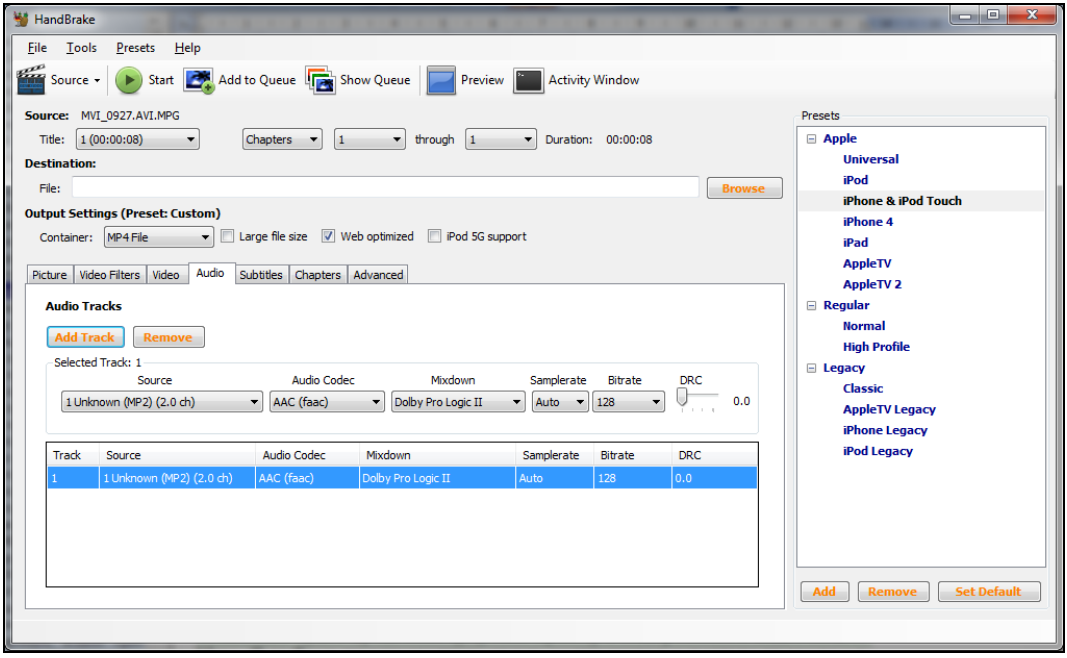


Рис. 5.26. Опции качества звука

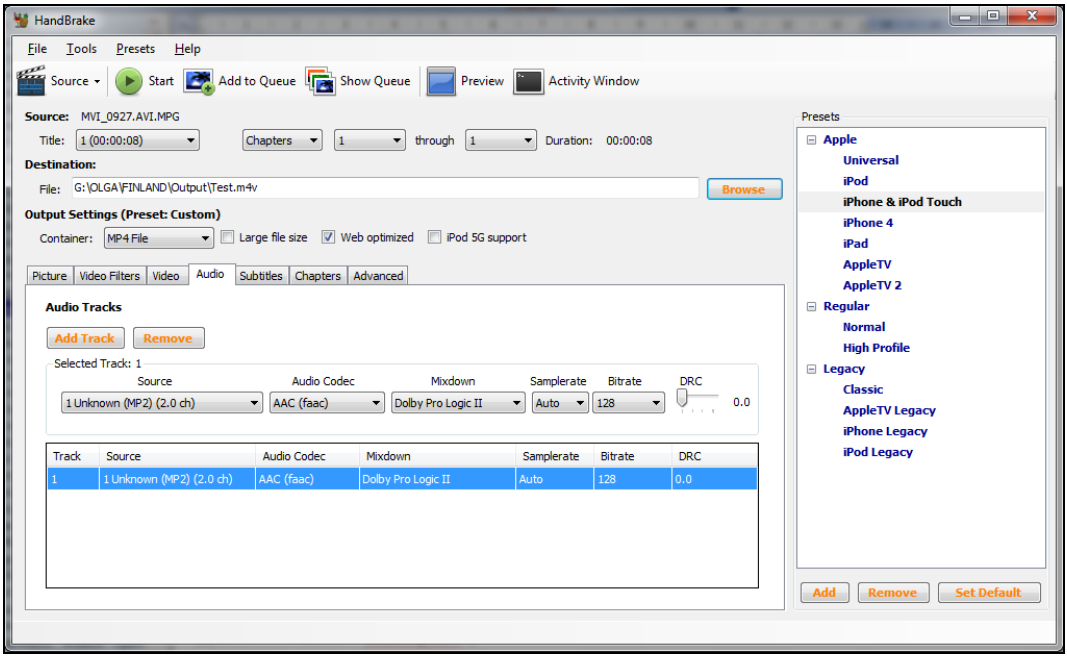


Рис. 5.27. Указание имени выводного файла

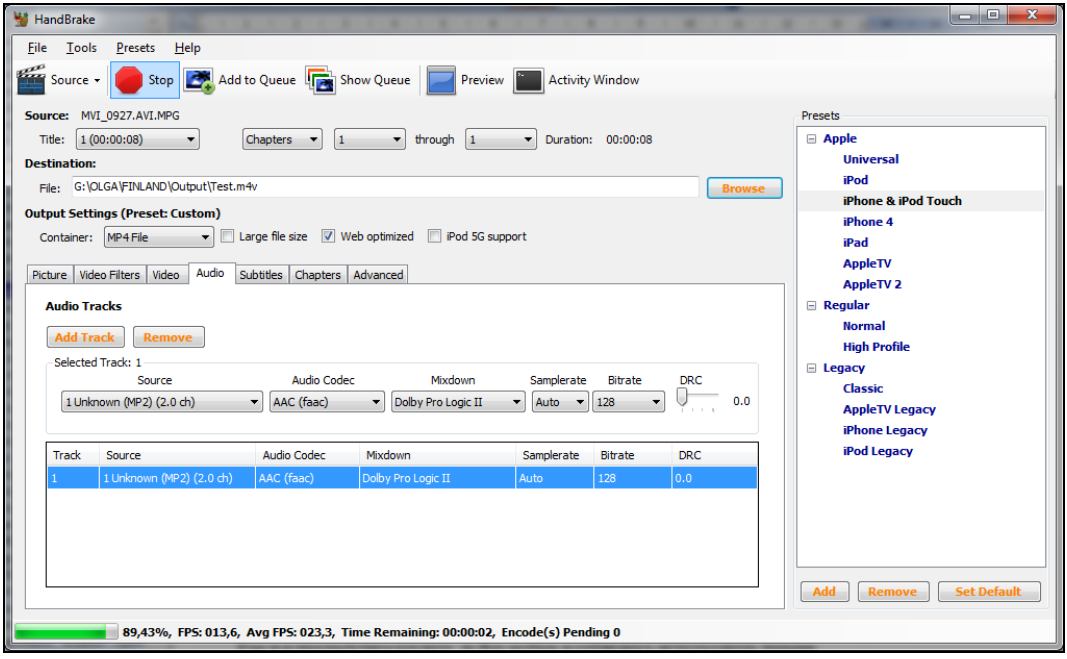


Рис. 5.28. Наберитесь терпения!

Наконец, щелкните по кнопке **Start**, чтобы начать кодирование.

HandBrake отобразит некоторые статистические данные, пока происходит кодирование видео (рис. 5.28).

Пакетное кодирование видео H.264 с помощью HandBrake

Как и в предыдущем разделе, в данном разделе я собираюсь использовать термин "видео H.264" как сокращение для термина "базовый профиль видео H.264 и упрощенный профиль аудио AAC в контейнере MPEG-4". Эта комбинация кодеков и контейнера будет иметь встроенную поддержку в Safari, Adobe Flash и на устройствах iPhone и Google Android.

HandBrake поставляется и в виде утилиты командной строки. Как и в случае с `ffmpeg2theora`, утилита командной строки HandBrake предлагает ошеломительный набор опций. Чтобы просмотреть их полный набор, введите из командной строки команду `HandBrakeCLI --help`. Я же опишу здесь лишь некоторые, наиболее употребительные:

- `--preset` *X*, где *X* — имя набора предопределенных опций HandBrake. Предопределенный набор опций, который понадобится вам для видео H.264, предназначенного для использования в Web, называется `"iPhone & iPod Touch"`, и обратите внимание, что все имя должно быть заключено в кавычки.

- `--width W`, где `W` — ширина кадра вашего закодированного видео. HandBrake автоматически настроит высоту кадра, чтобы сохранить пропорции оригинала.
- `--vb Q`, где `Q` представляет собой средний битрейт (измеряемый в килобитах в секунду).
- `--two-pass`, опция, которая активизирует двухпроходное кодирование.
- `--turbo`, опция, которая активизирует турборежим для первого прохода при двухпроходном кодировании.
- `--input F`, где `F` — имя файла вашего исходного видео.
- `--output E`, где `E` — имя выводного файла, который будет содержать ваше закодированое видео.

Пример, иллюстрирующий вызов HandBrake из командной строки, с использованием опций, соответствующих выбранным нами при использовании графической версии HandBrake, приведен в листинге 5.2.

Листинг 5.2. Вызов HandBrake из командной строки

```
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch"
                            --width 320
                            --vb 600
                            --two-pass
                            --turbo
                            --input pr6.dv
                            --output pr6.mp4
```

Эта команда запускает HandBrake с набором предопределенных опций `"iPhone & iPod Touch"`, изменяет размеры видеокадра до 320×240, устанавливает средний битрейт на 600 kbps, активизирует двухпроходное кодирование с первым проходом в режиме `turbo`, читает файл `pr6.dv` и кодирует его в файл `pr6.mp4`.

Кодирование видео WebM с помощью ffmpeg

WebM полностью поддерживается утилитой `ffmpeg 0.6` (**http://www.ffmpeg.org/**) и более поздними версиями. Из командной строки запустите команду `ffmpeg` без параметров и убедитесь в том, что она скомпилирована с поддержкой VP8, как показано в листинге 5.3.

Листинг 5.3. Запуск команды `ffmpeg` без параметров позволяет проверить, скомпилирована ли она с поддержкой VP8

```
you@localhost$ ffmpeg
FFmpeg version SVN-r23197, Copyright (c) 2000-2010 the FFmpeg developers
  built on May 19 2010 22:32:20 with gcc 4.4.3
  configuration: --enable-gpl --enable-version3 --enable-nonfree --enable-
postproc --enable-pthreads --enable-libfaac --enable-libfaad --enable-
libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-
libtheora --enable-libx264 --enable-libxvid --enable-x11grab --enable-
libvorbis --enable-libvpx
```

Если вы не увидите "магических слов" `--enable-libvorbis` и `--enable-libvpx`, это значит, что у вас — неподходящая версия `ffmpeg`. Если вы компилируете `ffmpeg` самостоятельно, проверьте, не установлено ли у вас две версии. Это нормально, они не будут конфликтовать друг с другом. Вам просто понадобится указать полный путь к версии `ffmpeg`, обеспечивающей поддержку VP8.

Я собираюсь выполнить двухпроходное кодирование. При первом проходе выполняется анализ входного видеофайла (`-i pr6.dv`) и ведется запись в журнал некоторой собранной статистики (файл журнала автоматически получит имя `pr6.dv-0.log`). Я указываю видеокодек с помощью параметра `-vcodec`, как показано в листинге 5.4.

Листинг 5.4. Запуск команды ffmpeg из командной строки с целью выполнения двухпроходного кодирования

```
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16  -keyint_min 0 -
g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 614400 -s
320x240 -aspect 4:3 -an -y NUL
```

Большая часть командной строки `ffmpeg` не имеет ничего общего с VP8 или WebM. Библиотека `libvpx` поддерживает ряд опций, специфичных для VP8, которые вы можете передать `ffmpeg`, но я пока не знаю, будут ли какие-либо из них работать. Когда я найду удовлетворительное объяснение этих опций, я включу их в следующее издание этой книги.

На втором проходе `ffmpeg` прочтет статистику, записанную на предыдущем проходе, и фактически выполнит кодирование видео и аудио. На выходе вы получите созданный ею файл `.webm` (листинг 5.5).

Листинг 5.5. На втором проходе вы на выходе получите видеофайл, созданный `ffmpeg`

```
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16  -keyint_min 0 -
g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 614400 -s
320x240 -aspect 4:3 -acodec libvorbis -y pr6.webm
```

Эта команда имеет пять важных параметров:

- `-vcodec libvpx` — указывает, что мы кодируем с использованием видеокодека VP8. WebM всегда использует видео VP8.
- `-b 614400` — указывает битрейт. В отличие от остальных форматов, `libvpx` ожидает, что вы укажете битрейт в фактическом количестве бит, а не килобит. Если вы хотите получить видео с битрейтом 600 kbps, умножьте 600 на 1024, и вы получите значение 614400.
- `-s 320x240` — указывает размер кадра для целевого видео, ширину и высоту.
- `-aspect 4:3` — указывает пропорции видеокадра в целевом файле. Видео стандартного разрешения обычно имеет пропорции 4:3, но большинство видеофайлов высокого разрешения (high-definition video) используют другие пропорции: 16:9 или 16:10. В результате тестирования я определил, что эти пропорции

лучше указывать явно через командную строку, а не полагаться на то, что `ffmpeg` определит их автоматически.

- `-acodec libvorbis` — указывает, что мы собираемся кодировать с использованием аудиокодека Vorbis. WebM всегда использует аудиокодек Vorbis.

Наконец, перейдем к разметке

Я абсолютно уверен в том, что этот материал должен присутствовать в любой книге, посвященной HTML. Так где же у нас разметка?

HTML5 предоставляет вам два пути включить видео в ваши Web-страницы. Оба метода связаны с использованием элемента `<video>`. Если у вас есть только один видеофайл, вы можете просто поставить на него ссылку в атрибуте `src`. Этот подход имеет замечательное сходство с включением изображения при помощи тега `<img src="...">`.

Включение одного видеофайла

```
<video src="pr6.webm"></video>
```

С технической точки зрения это все, что вам требуется. Но, как и в случае с тегом `<img>`, вам всегда следует добавлять атрибуты `width` и `height` в ваши теги `<video>`. Атрибуты ширины и высоты могут быть такими же, как и максимальные ширина и высота, которые вы указали в процессе кодирования. Не беспокойтесь, если один из размеров окажется чуть меньше максимального. Ваш браузер отцентрирует центр видеокадра по центу прямоугольника, заданного тегом `<video>`. Пропорции кадра при этом искажены не будут.

```
<video src="pr6.webm" width="320" height="240"></video>
```

По умолчанию, элемент `<video>` не будет отображать никаких элементов управления воспроизведением. Вы можете самостоятельно создать их с помощью старых добрых HTML, CSS и JavaScript. Элемент `<video>` имеет методы наподобие `play()` и `pause()`[1], а также перезаписываемое свойство, называющееся `currentTime` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#dom-media-currenttime**). Кроме того, имеются перезаписываемые свойства для управления громкостью записи и воспроизведения `volume` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#dom-media-volume**), а также свойства воспроизведения без звука `muted` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#dom-media-muted**). Поэтому в вашем распоряжении имеется все, чтобы построить собственный интерфейс.

Если построение собственного интерфейса не входит в ваши планы, вы можете дать браузеру указание отобразить встроенный набор элементов управления. Чтобы сделать это, достаточно включить атрибут `controls` в ваш тег `<video>`.

```
<video src="pr6.webm" width="320" height="240" controls></video>
```

Существуют еще два необязательных элемента управления, которые я хотел бы упомянуть, прежде чем переходить к дальнейшему изложению материала: `preload`

[1] См. **http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#playing-the-media-resource**.

и `autoplay`. Не стоит возмущаться, лучше дайте мне объяснить, почему они полезны. Атрибут `preload` сообщает браузеру, что вы хотели бы, чтобы он начал загрузку видео сразу же, как только начнется загрузка страницы. Это имеет смысл делать, если основная цель вашей страницы — просмотр видео. С другой стороны, если видео — это только вспомогательный материал, и просматривать его будут немногие посетители, тогда атрибуту `preload` можно назначить значение `none`, чтобы браузер минимизировал сетевой трафик.

Рассмотрим пример видео, которое начинает загружаться (но не воспроизводиться) при загрузке страницы:

```
<video src="pr6.webm" width="320" height="240" preload></video>
```

Рассмотрим пример видео, которое не начнет загружаться сразу же при начале загрузки страницы:

```
<video src="pr6.webm" width="320" height="240" preload="none"></video>
```

Атрибут `autoplay` делает в точности то, о чем говорит его название: он сообщает браузеру о том, что вы хотели бы начать загрузку видеофайла сразу же, когда загрузится страница, и что вы хотите как можно скорее начать автоматическое воспроизведение видео. Некоторым людям это нравится; но некоторые этого просто терпеть не могут. Но позвольте мне объяснить, почему в HTML5 важно иметь атрибут, подобный этому. Некоторым людям хочется, чтобы их видеофайлы воспроизводились автоматически, даже если это и раздражает посетителей. Если в HTML5 не был определен стандартный способ автоматического воспроизведения видео, люди начали бы прибегать к хакам на JavaScript, чтобы сделать это в любом случае. Например, это можно сделать, вызвав метод видео `play()` во время события загрузки окна. Посетителям стало бы намного сложнее оказывать противодействие этому трюку. С другой стороны, это — всего лишь вопрос добавления расширения к вашему браузеру (или, при необходимости, написания собственного расширения), чтобы сказать "игнорируйте атрибут `autoplay`, я хочу, чтобы видеофайлы никогда не воспроизводились автоматически".

Вот пример видео, которое начнет загружаться и воспроизводиться как можно скорее после загрузки страницы:

```
<video src="pr6.webm" width="320" height="240" autoplay></video>
```

В листинге 5.6 представлен скрипт Greasemonkey (**http://www.greasespot.net/**), который вы можете установить на своей локальной копии Firefox. Этот скрипт предотвратит автоматическое воспроизведение видео HTML5. Он использует атрибут HTML5, называющийся `autoplay` и определенный в DOM. Этот атрибут представляет собой эквивалент JavaScript для атрибута `autoplay` в вашей разметке HTML. Минимизированный код этого скрипта доступен здесь: **http://diveintohtml5.org/examples/disable_video_autoplay.user.js**.

Листинг 5.6. Скрипт `disable_video_autoplay.user.js` предотвратит автоматическое воспроизведение видео HTML5

```
// ==UserScript==
// @name           Блокировка автоматического воспроизведения видео
// @namespace      http://diveintomark.org/projects/greasemonkey/
```

```
// @description    Скрипт блокирует автоматическое воспроизведение
//                 элементов видео HTML5
// @include        *
// ==/UserScript==

var arVideos = document.getElementsByTagName('video');
for (var i = arVideos.length - 1; i >= 0; i--) {
    var elmVideo = arVideos[i];
    elmVideo.autoplay = false;
}
```

Но подождите… Если вы читаете эту главу, то вы уже знаете, что у вас не один видеофайл, а целых три. Один из них — это файл `.ogv`, который вы создали с помощью Firefogg или `ffmpeg2theora`. Второй видеофайл — это файл `.mp4`, созданный с помощью HandBrake. Наконец, третий файл — это `.webm`-файл, который вы создали с помощью `ffmpeg`. HTML5 предоставляет способы ссылаться на все три типа видеофайлов: элемент `<source>`. Каждый элемент `<video>` может содержать более одного элемента `<source>`. Ваш браузер просмотрит список видеоисточников по порядку и воспроизведет первый из них, который он способен воспроизвести.

Это поднимает еще один вопрос: как браузер узнает о том, какой видеофайл он может воспроизвести? Хорошо, пусть в наихудшем сценарии он будет загружать все видеофайлы по очереди и пытаться их воспроизводить. Это гигантский, непроизводительный расход полосы пропускания. Вы серьезно сэкономите на сетевом трафике, если проинформируете браузер о его возможностях до загрузки каждого из видеофайлов. Это можно сделать с помощью атрибута `type` в элементе `<source>`.

Все это в совокупности выглядит так, как показано в листинге 5.7.

Листинг 5.7. Разметка, использующая атрибут `type` в элементе `<source>`, информирующая браузер о его возможностях до загрузки видеофайлов

```
<video width="320" height="240" controls>
  <source src="pr6.mp4"  type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
  <source src="pr6.ogv"  type='video/ogg; codecs="theora, vorbis"'>
</video>
```

Давайте разобьем этот код на фрагменты и рассмотрим их по отдельности. Элемент `<video>` указывает ширину и высоту видео, но не ссылается на видеофайл. Внутри элемента `<video>` находятся три элемента `<source>`. Каждый элемент `<source>` ссылается на один видеофайл (с помощью атрибута `src`) и предоставляет информацию о видеоформате (с помощью атрибута `type`).

Атрибут `type` выглядит сложным — и не просто выглядит, он и в самом деле чертовски сложен. Он представляет собой комбинацию информации трех типов: контейнерный формат, видеокодек и аудиокодек. Начнем с самого нижнего уровня. Для видеофайла `.ogv` контейнерным форматом будет формат Ogg, представленный здесь как `video/ogg`. Говоря технически, это — тип MIME для видеофайлов Ogg. В качестве видеокодека используется Theora, а в качестве аудиокодека — Vorbis. Это достаточно просто, за исключением того, что формат значения атрибута выглядит несколько мудреным. Само значение должно содержать символы кавычек, а это значит, что вы должны будете использовать разные типы кавычек, в которые заключается каждое значение.

```
<source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"'>
```

WebM выглядит практически так же, но имеет другой тип MIME (`video/webm` вместо `video/ogg`) и другой видеокодек (`vp8` вместо `theora`), который указывается в параметре `codecs`.

```
<source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
```

Видео H.264 выглядит еще сложнее. Вспомните, что когда я говорил о видео H.264, я упоминал, что и для видео H.264, и для аудио AAC существуют различные "профили"? В примере, который рассматривался в этой главе, мы использовали для кодирования "базовый" профиль H.264 и "упрощенный" (low-complexity), упаковывая все это в контейнер MPEG-4. Вся эта информация включена в атрибут `type`.

```
<source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

Преимущество, которое мы получили за счет всей этой возни, заключается в том, что браузер сначала проверит атрибут `type`, чтобы проверить, а может ли он воспроизводить конкретный видеофайл. Если браузер определяет, что он не может воспроизвести конкретный видеофайл, то он не будет загружать этот файл. Вот именно — ни одна часть этого файла не будет скачиваться только для того, чтобы убедиться в невозможности ее воспроизведения. За счет этого вы сэкономите на полосе пропускания, а ваши посетители быстрее просмотрят то видео, которое они желали просмотреть.

Если для кодирования ваших видеофайлов вы будете следовать инструкциям, приведенным в данной главе, то вы можете просто копировать и вставлять значения атрибутов `type` из рассматриваемого примера. В противном случае вам потребуется разработать значения параметров `type` самостоятельно (**http://wiki.whatwg.org/wiki/Video_type_parameters**).

Замечание Профессора Разметки

На момент написания этих строк (20 мая 2010), iPad, работавшие под управлением iOS 3.*x*, имели баг, который не позволял устройству "замечать" что бы то ни было, кроме первого из перечисленных видеофайлов. iOS 4 (бесплатное обновление для всех устройств iPads) исправляет этот баг. Но если вы хотите предоставлять видео всем владельцам iPad, включая тех, кто еще не выполнил апгрейда, то вам нужно составлять список так, чтобы MP4-файл был в нем первым, и только затем перечислять файлы свободных видеоформатов.

Типы MIME проявляют свой скверный "характер"

Головоломка под названием "видео" состоит из такого количества фрагментов, что я сначала даже колебался — стоит ли вообще даже затрагивать этот вопрос. Однако эта тема очень важна, потому что неправильно сконфигурированный Web-сервер может довести вас до отчаяния, когда вы будете пытаться выяснить, почему ваши видеофайлы воспроизводятся на локальном компьютере, но не воспроизводятся, когда вы распространяете их на ваш "производственный" сайт. Если вы столкнетесь с этой проблемой, то возможно, что основной причиной окажутся типы MIME.

Я упоминал типы MIME в главе, посвященной истории развития HTML, но вполне возможно, что вы читали ее недостаточно внимательно и пропустили этот материал, недооценив его важности. Поэтому давайте вернемся к этому вопросу.

Крик души Профессора Разметки

ВИДЕОФАЙЛЫ ДОЛЖНЫ ПОСТАВЛЯТЬСЯ С НАДЛЕЖАЩИМИ ТИПАМИ MIME!

Хорошо, а что такое "надлежащий тип MIME"? Вы его уже видели; он является частью значения атрибута `type` в элементе `<source>`. Но установки атрибута `type` в вашей разметке HTML недостаточно. Вам, помимо прочего, необходимо еще гарантировать, что ваш Web-сервер включает надлежащий тип MIME в заголовок HTTP `Content-Type`.

Если вы используете Web-сервер Apache или некоторую производную от Apache, то вы можете использовать директиву `AddType` (**http://httpd.apache.org/docs/2.0/mod/mod_mime.html#addtype**) в вашем общесистемном файле `httpd.conf` или в файле `.htaccess` в том каталоге, в котором вы храните свои видеофайлы (листинг 5.8). Если вы используете какой-либо другой Web-сервер, сверьтесь с документацией к вашему серверу, чтобы выяснить, как следует устанавливать заголовки HTTP `Content-Type` для конкретных типов файлов.

Листинг 5.8. Установка типов MIME в заголовки HTTP на сервере Apache

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

Первая строка предназначена для видео в контейнере Ogg. Вторая строка предназначена для видео в контейнере MPEG-4. Третья строка предназначена для WebM. Установите эти настройки один раз, после чего вы сможете забыть о них навсегда. Если вы забудете их установить, то некоторые браузеры обязательно откажутся воспроизводить ваши файлы, несмотря даже на то, что вы включите типы MIME в атрибут `type` в вашей разметке HTML.

Еще больше "кровавых" подробностей борьбы с настройками Web-сервера можно найти в следующей отличной статье Центра разработки Mozilla (Mozilla Developer Center): *"Configuring servers for Ogg media"* (**https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media**). Советы, приведенные в этой статье, подходят также и для видео в форматах MP4 и WebM.

Как обстоят дела с IE?

Новейшая на текущий момент версия Internet Explorer Release Candidate поддерживает элемент HTML5 `<video>` (**http://msdn.microsoft.com/en-us/ie/ff468705.aspx#_HTML5_video_audio**), но компания Microsoft публично дала обещание, что окончательная версия IE 9 будет поддерживать видео H.264 и аудио AAC в контейнере MPEG-4, в точности так, как это реализовано в браузере Safari и устройствах iPhone (**http://blogs.msdn.com/b/ie/archive/2010/03/16/html5-hardware-accelerated-first-ie9-platform-preview-available-for-developers.aspx**).

Хорошо, а как обстоят дела с более старыми версиями Internet Explorer? Под "более старыми" версиями подразумеваются все версии этого браузера, находящиеся в употреблении по сегодняшний день, включая IE 8. Большинство пользователей, которые на сегодняшний день работают с Internet Explorer, как правило, устанавливают также и плагин Adobe Flash. Современные версии Adobe Flash (начиная с версии 9.0.60.184) поддерживают видео H.264 и аудио AAC в контейнере MPEG-4, в точности так же, как это реализовано в Safari и iPhone. Как только вы закодируете видео H.264 для Safari, вы сможете воспроизвести его и на любом видеопроигрывателе на основе Flash, если вы обнаружите, что кто-то из ваших посетителей все еще пользуется браузером, не обеспечивающим поддержки HTML5.

FlowPlayer (**http://flowplayer.org/**) — это Flash-видеопроигрыватель на основе открытого кода, поставляемый по лицензии GPL. Доступны, в том числе, и коммерческие лицензии (**http://flowplayer.org/download/**). FlowPlayer ничего не "знает" об элементе `<video>`. Он не будет каким-то "магическим" образом преобразовывать тег `<video>` в объект Flash. Но спецификация HTML5 очень хорошо продумана и позволяет решать проблемы наподобие этой, потому что вы можете встроить в элемент `<video>` элемент `<object>`. Браузеры, которые не поддерживают видео HTML5, будут игнорировать элемент `<video>`, и вместо этого будут визуализировать встроенный элемент `<object>`, что активизирует плагин Flash и приведет к воспроизведению видео через FlowPlayer. Браузеры, которые поддерживают видео HTML5, найдут видеофайл, который они могут воспроизвести, и воспроизведут его, игнорируя встроенный элемент `<object>`.

Последняя особенность представляет собой ключ ко всей головоломке: HTML5 указывает, что все элементы (отличные от элементов `<source>`) являются дочерними элементами для элемента `<video>`, и поэтому должны быть проигнорированы. Это позволяет использовать элементы видео в формате HTML5 в новых браузерах, а если пользователь работает со старым браузером, то в качестве резервной возможности полагаться на Flash, без необходимости прибегать к причудливым "хакам" JavaScript. Подробнее об этом подходе можно прочесть в статье *"Video For Everybody"* (**http://camendesign.com/code/video_for_everybody**).

Вопросы, касающиеся устройств iPhone и iPad

iOS — это операционная система компании Apple, управляющая работой устройств iPhone, iPod Touch, и iPad. Операционная система iOS 3.2 имеет некоторые проблемы с воспроизведением видео в формате HTML5.

1. iOS не распознает видео, если вы включите атрибут `poster`. Атрибут `poster` элемента `<video>` позволяет вам отображать индивидуальные изображения в течение того времени, пока происходит загрузка видео, или до тех пор, пока пользователь не щелкнет по кнопке начала воспроизведения. Этот баг был исправлен в iOS 4.0, но должно пройти время, чтобы все пользователи перешли на использование новой версии ОС.
2. Если у вас имеется множество элементов `<source>`, iOS будет распознавать только первый элемент из списка. Так как устройства, работающие под управлением iOS, поддерживают только комбинацию H.264+AAC+MP4, это означает, что MP4-файлы всегда нужно указывать первыми в списке. Этот баг тоже исправлен в iOS 4.0, но многие пользователи еще работают со старыми версиями iOS, и должно пройти определенное время, прежде чем все пользователи обновят свои операционные системы до новой версии.

Проблемы с устройствами Android

Android — это операционная система Google, под управлением которой работает целый ряд различных смартфонов и мобильных устройств. ОС Android версий более ранних, чем 2.3, имела ряд проблем с воспроизведением видео в формате HTML5.

1. Атрибут `type` для элементов `<source>` приводил Android в замешательство. Забавно, что единственный метод, позволявший добиться, чтобы эта ОС начала распознавать источник видео, заключается в том, чтобы полностью пропустить атрибут `type` и гарантировать, что имя файла, представляющего собой комбинацию H.264+AAC+MP4, обязательно имел расширение `.mp4`. Очевидно, что это не влияет на способность любого другого браузера к обнаружению поддержки видео; при отсутствии атрибута `type`, остальные браузеры, по всей вероятности, тоже строят свои предположения на основе расширения имени файла. Тем не менее, вы можете использовать атрибут `type` для видеофайлов всех остальных типов. Этот баг был исправлен в версии Android 2.3.
2. Атрибут `controls` не поддерживается. Никаких неприятных побочных эффектов от его включения не наблюдается, но Android все равно не будет отображать никаких элементов пользовательского интерфейса для управления воспроизведением видео. Если вы хотите, чтобы пользователь мог управлять процессом воспроизведения видео, вы должны будете предоставить собственные интерфейсные элементы управления. Как минимум, вам необходимо предоставить скрипт, который запускает воспроизведение видео, когда пользователь выполняет по нему щелчок мышью. Этот баг тоже был исправлен в версии Android 2.3.

Полноценный, "живой" пример

Рассмотрим "живой" пример видео, который использует описанные в данной главе приемы. Я взял за основу пример кода из статьи "Video For Everybody" (**http://camendesign.com/code/video_for_everybody**) и дополнил его кодированием видео в формате WebM. Один и тот же исходный файл я закодировал в три различных формата с помощью команд, перечисленных в листинге 5.9.

Листинг 5.9. Пример, кодирующий видео в три различных формата

```
## Theora/Vorbis/Ogg
you@localhost$ ffmpeg2theora --videobitrate 200 --max_size 320x240 --output
pr6.ogv pr6.dv


## H.264/AAC/MP4
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch" --vb 200 --width
320 --two-pass --turbo --optimize --input pr6.dv --output pr6.mp4


## VP8/Vorbis/WebM
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16  -keyint_min 0 -
g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 204800 -s
320x240 -aspect 4:3 -an -f webm -y NUL
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16  -keyint_min 0 -
g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 204800 -s
320x240 -aspect 4:3 -acodec libvorbis -ac 2 -y pr6.webm
```

Окончательная разметка использует элемент `<video>` для видео в формате HTML5, встроенный элемент `<object>` для тех случаев, когда пользователям старых браузеров приходится полагаться на Flash, и небольшой скрипт для удобства пользователей устройств Android, как показано в листинге 5.10.

Листинг 5.10. Окончательный вариант разметки, использующей комбинацию HTML5 и Flash

```
<video id="movie" width="320" height="240" preload controls>
  <source src="pr6.mp4" />
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"' />
  <object width="320" height="240" type="application/x-shockwave-flash"
    data="flowplayer-3.2.1.swf">
    <param name="movie" value="flowplayer-3.2.1.swf" />
    <param name="allowfullscreen" value="true" />
    <param name="flashvars" value='config={"clip": {"url":
"http://wearehugh.com/dih5/good/bbb_480p.mp4", "autoPlay":false, "autoBuffer-
ing":true}}' />
```

```
    <p>Download video as <a href="pr6.mp4">MP4</a>, <a
href="pr6.webm">WebM</a>, or <a href="pr6.ogv">Ogg</a>.</p>
  </object>
</video>
<script>
  var v = document.getElementById("movie");
  v.onclick = function() {
    if (v.paused) {
      v.play();
    } else {
      v.pause();
    }
  });
</script>
```

Эта комбинация HTML5 и Flash должна помочь вам просматривать видео практически в любом браузере и на любом устройстве.

Рекомендуемые материалы для дальнейшего чтения

- Материалы об элементе `<video>` из спецификации HTML5 — **http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#video**
- Video for Everybody — **http://camendesign.com/code/video_for_everybody**
- Популярное введение в кодирование видео — **http://diveintomark.org/tag/give**
- Theora 1.1 is released — what you need to know (материалы с сайта разработки Mozilla) — **http://hacks.mozilla.org/2009/09/theora-1-1-released/**
- Configuring servers for Ogg media (статья с центра разработки Mozilla) — **https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media**
- Encoding with the x264 codec — **http://www.mplayerhq.hu/DOCS/HTML/en/menc-feat-x264.html**
- Video type parameters — **http://wiki.whatwg.org/wiki/Video_type_parameters**
- Everything you need to know about HTML5 audio and video — **http://dev.opera.com/articles/view/everything-you-need-to-know-about-html5-video-and-audio/**
- Making HTML5 video work on Android phones — **http://www.broken-links.com/2010/07/08/making-html5-video-work-on-android-phones/**

Готовые сборки индивидуальных элементов управления для видео в формате HTML5:

- **http://videojs.com/**
- **http://mediaelementjs.com/**
- **http://www.kaltura.org/project/HTML5_Video_Media_JavaScript_Library**

Глава 6

Вы находитесь здесь (как и все остальные)

Геопозиционирование (geolocation) — это искусство определять свое географическое местоположение и (при желании) предоставлять эту информацию тем людям, которым вы доверяете. Существует множество способов определять ваше географическое местоположение, в том числе — ваш IP-адрес, ваше беспроводное сетевое соединение, вышка сотовой связи, с которой поддерживает "диалог" ваш мобильный телефон, или специализированное оборудование GPS, которое вычисляет ваши географические координаты на основе информации, принимаемой от навигационных спутников.

Вопрос к Профессору Разметки

В: Само слово "геопозиционирование" звучит пугающе. Могу ли я отключить эту функцию?

О: Конфиденциальность и "приватность" (privacy) — это совершенно очевидная проблема, когда речь заходит о предоставлении информации о вашем физическом местоположении на удаленный Web-сервер. Интерфейс прикладного программирования для геопозиционирования (geolocation API) явным образом утверждает: "Пользовательские агенты не должны отсылать на Web-сайты информацию о географическом местоположении без явного разрешения пользователя" (**http://www.w3.org/TR/geolocation-API/#security**). Иными словами, предоставление информации о вашем географическом местоположении кому бы то ни было всегда является делом добровольным. Если вы не хотите этого делать, вы этого делать не обязаны.

API геопозиционирования (Geolocation API)

API геопозиционирования (**http://www.w3.org/TR/geolocation-API/**) позволяет пользователю предоставлять информацию о своем географическом местоположении на те Web-сайты, которым он доверяет. Значения широты и долготы доступны для JavaScript на странице, которая, в свою очередь, может отсылать эти данные удаленному Web-серверу и выполнять различные операции над информацией о географических координатах, например, находить для вас местные предприятия или организации либо просто показывать вам ваше местоположение на карте.

Как видно из табл. 6.1, geolocation API поддерживается большинством браузеров на настольных компьютерах и мобильных устройствах. В дополнение к этому, некоторые более старые браузеры и устройства могут поддерживаться библиотеками-"обертками", как мы и увидим далее в этой главе.

Таблица 6.1. *Поддержка API геопозиционирования современными браузерами и мобильными устройствами*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
9.0+	3.5+	5.0+	5.0+	10.6+	3.0+	2.0+

Наряду с поддержкой стандартного API геопозиционирования, существует множество интерфейсов, специфичных для конкретных устройств на различных мобильных платформах. Я рассмотрю эту тему чуть далее в этой главе.

Продемонстрируйте мне код

API геопозиционирования полностью строится "вокруг" нового свойства глобального объекта `navigator`: это свойство `navigator.geolocation`.

Простейший вариант использования API геопозиционирования показан в листинге 6.1.

Листинг 6.1. Простейший вариант использования API геопозиционирования

```
function get_location() {
  navigator.geolocation.getCurrentPosition(show_map);
}
```

Этот вариант не предполагает выявления поддержки браузером, не имеет никакой обработки ошибок и никаких опций. Скорее всего, ваше Web-приложение должно будет реализовать, как минимум, первые два пункта из перечисленных трех. Чтобы определить, поддерживается ли вашим браузером или мобильным устройством API геопозиционирования, вам следует использовать библиотеку `Modernizr` (**http://www.modernizr.com/**), как показано в листинге 6.2.

Листинг 6.2. Использование библиотеки `Modernizr` для выявления поддержки API геопозиционирования

```
function get_location() {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(show_map);
  } else {
    // no native support; maybe try Gears?
  }
}
```

Что вам следует делать в случае отсутствия поддержки геопозиционирования, вы решаете сами. Совсем скоро я объясню опцию Gears, на которую можно положиться в случае отсутствия такой поддержки, но сначала я хотел бы поговорить о том, что происходит во время вызова к функции `getCurrentPosition()`. Как я

уже отмечал в начале этой главы, поддержка геопозиционирования — дело добровольное. Это означает, что ваш браузер никогда не будет принуждать вас к тому, чтобы вы раскрыли ваше текущее географическое местоположение удаленному серверу. Пользовательский интерфейс зависит от используемого браузера. В Mozilla Firefox вызов функции `getCurrentPosition()` из API геопозиционирования приведет к всплытию "информационной панели" (infobar) в верхней части окна браузера. Эта информационная панель показана на рис. 6.1.

Рис. 6.1. Информационная панель геопозиционирования

В информационной панели происходит множество интересных событий. Вы, как конечный пользователь:

- получаете уведомление о том, что Web-сайт запрашивает информацию о вашем географическом местоположении;
- получаете информацию о том, *какой W*eb-сайт заинтересован в сведениях о вашем географическом местоположении;
- получаете возможность пролистать страницу справочной системы Mozilla "Location-Aware Browsing" (**http://www.mozilla.com/en-US/firefox/geolocation/**), где дается объяснение всем происходящим событиям; Google предоставляет информацию о вашем географическом местоположении и хранит ее в соответствии со своей политикой конфиденциальности (**http://www.google.com/intl/en/privacy/lsf.html**);
- получаете возможность принять решение о том, следует ли предоставлять в общий доступ информацию о вашем географическом местоположении;
- можете отказаться предоставлять информацию о вашем географическом местоположении;
- можете сообщить вашему браузеру о необходимости запомнить ваш выбор (в любом случае — поставлять информацию или не делать этого), так чтобы никогда больше не видели этой информационной панели на данном Web-сайте.

Далее, эта информационная панель является:

- немодальной, так что она не помешает вам переключиться на другое окно браузера или перейти на другую вкладку;
- специфичной для вкладок, так что она исчезнет, если вы переключитесь на другое окно браузера или вкладку, и вновь появится, когда вы переключитесь обратно на исходную вкладку;
- безусловной, так что Web-сайт не сможет ее обойти;
- блокирующей, так что Web-сайт не будет иметь шансов определить ваше географическое местоположение, пока он ожидает вашего ответа.

Вы только что просмотрели код JavaScript, который приводит к появлению информационной панели. Это — единственный функциональный вызов, который принимает функцию обратного вызова (которую я назвал `show_map`). Вызов к `getCurrentPosition()` вернется немедленно, но это не значит, что вы получите доступ к местоположению пользователя. Единственный раз, когда вы гарантиро-

ванно имеете информацию о географическом местоположении — функция обратного вызова. Функция обратного вызова выглядит так, как показано в листинге 6.3.

Листинг 6.3. Функция обратного вызова для получения информации о географическом местонахождении пользователя

```
function show_map(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  // Теперь нужно отобразить карту или выполнить еще что-нибудь интересное!
}
```

Функция обратного вызова будет вызвана с единственным параметром — объектом, обладающим двумя свойствами: `coords` и `timestamp`. Свойство `timestamp` представляет собой именно то, о чем говорит его название: дату и время, на тот момент, когда производилось определение географического местоположения. Поскольку все это происходит асинхронно, вы не можете заранее узнать, когда это произойдет. Возможно, пользователю потребуется некоторое время, чтобы увидеть информационную панель и согласиться предоставить свои координаты. Устройства, имеющие специализированное оборудование GPS, тоже могут нуждаться во времени на соединение со спутником GPS, и так далее. Объект `coords` обладает такими свойствами, как `latitude` и `longitude`, которые указывают именно то, о чем говорят их названия — географические координаты пользователя (широту и долготу). Полный список свойств объекта `coords`, описывающего местоположение пользователя в реальном мире, приведен в табл. 6.2.

***Таблица 6.2.** Свойства объекта coords*

Свойство	Тип	Примечание
`coords.latitude`	`double`	десятичные градусы
`coords.longitude`	`double`	десятичные градусы
`coords.altitude`	`double` или `null`	метры над референц-**эллипсоидом (https**://secure.wikimedia.org/wikipedia/en/wiki/Reference_ellipsoid, **http://tinyurl.com/4nyqr4n**)
`coords.accuracy`	`double`	метры
`coords.altitudeAccuracy`	`double` или `null`	метры
`coords.heading`	`double` или `null`	градусы по часовой стрелке, отсчитываемые от истинного направления северного меридиана (**https://secure.wikimedia.org/wikipedia/en/wiki/True_north**)
`coords.speed`	`double` или `null`	метры в секунду
`timestamp`	`DOMTimeStamp`	как объект `Date()`

Гарантируется наличие только трех свойств (`coords.latitude`, `coords.longitude` и `coords.accuracy`). Остальные свойства могут быть обнулены, в зависимости от возможностей вашего устройства и от сервера, с которым оно поддерживает связь. Свойства направления и скорости вычисляются на основе предыдущего местоположения пользователя, если это возможно.

Обработка ошибок

Геопозиционирование — процесс сложный, поэтому очень многие вещи могут пойти "не так". Я уже упоминал аспект "согласие пользователя". Если вашему Web-приложению требуются географические координаты пользователя, но пользователь не желает их предоставлять, то ничего не поделаешь. Пользователь всегда прав. Но как все это выглядит в коде программы? Выглядит это как второй аргумент `getCurrentPosition()` — функции обратного вызова для обработки ошибок (листинг 6.4).

Листинг 6.4. Функция обратного вызова имеет второй аргумент для обработки ошибок

```
navigator.geolocation.getCurrentPosition(
  show_map, handle_error)
```

Если что-то пойдет "не так", ваша функция обратного вызова будет вызвана с объектом `PositionError`, свойства которого кратко перечислены в табл. 6.3.

Таблица 6.3. *Объект* `PositionError`

Свойство	Тип	Замечания
`code`	`short`	Значение, полученное путем энумерации
`message`	`DOMString`	Не предназначается для конечных пользователей

Свойство `code` принимает одно из следующих значений:

- `PERMISSION_DENIED (1)`, если пользователь щелкнет по кнопке **Don't Share** или каким-либо иным способом откажется предоставить информацию о своих географических координатах;
- `POSITION_UNAVAILABLE (2)`, если сеть не работает или нет возможности установить контакт со спутниками;
- `TIMEOUT (3)` — если сеть работает, но вычисление географических координат пользователя требует слишком длительного времени. Что значит "требует слишком длительного времени"? Это я объясню вам в следующем разделе;
- `UNKNOWN_ERROR (0)` — если имеет место какая-то другая ошибка.

При невозможности получения географических координат пользователя необходимо корректно завершить работу. Как это делается, показано в листинге 6.5.

Листинг 6.5. Корректное завершение работы при невозможности получения координат пользователя

```
function handle_error(err) {
  if (err.code == 1) {
    // пользователь ответил отказом!
  }
}
```

Вопрос к Профессору Разметки

В: Работает ли API геопозиционирования на Международной космической станции, на Луне или на других планетах?

О: Определение API геопозиционирования явно утверждает: "Система географических координат, используемая атрибутами API геопозиционирования — это мировая геодезическая система (World Geodetic System, 2d), см. **http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf**. Никакие другие относительные системы не поддерживаются" (**http://www.w3.org/TR/geolocation-API/#coordinates_interface**). Международная космическая станция (International Space Station) находится на земной орбите, так что астронавты (**https://twitter.com/Astro_TJ**), находящиеся на ее борту, могут описывать свое местоположение, указывая широту, долготу и высоту. Однако Мировая геодезическая система привязана к началу координат, совпадающему с центром Земли, так что ее нельзя использовать на Луне и других планетах.

Выбор! Мне нужна возможность выбора!

Некоторые популярные мобильные устройства — например, iPhone и телефоны Android — поддерживают два метода вычисления вашего местоположения. Первый способ определяет ваше местоположение, пользуясь методом триангуляции на основании вашего относительного расстояния до различных вышек сотовой связи, управляемых вашим сотовым оператором. Это — первый из известных методов. Он очень быстр и не требует использования специализированного оборудования GPS. Однако данный метод дает вам лишь приблизительное понятие о вашем фактическом местоположении. В зависимости от того, какое количество вышек сотовой связи имеется в вашей области, "приблизительный район" вашего местоположения может составлять от небольшого городского квартала до квадрата "по несколько километров в каждом направлении".

Второй способ использует специализированное GPS-оборудование, выделенное специально для того, чтобы поддерживать коммуникации с навигационными спутниками, вращающимися на околоземных орбитах. Обычно GPS-системы могут определять ваше географическое местоположение с точностью до метров.

Крупным недостатком таких устройств является их высокое энергопотребление, в результате чего устройства отключаются в самый неподходящий момент (еще до того, как вам реально потребуется ими воспользоваться). Это значит, что пусковая задержка будет иметь место всегда — она нужна на инициализацию соединения между вашим устройством и спутником GPS. Если вы когда-либо пользовались сервисом Google Maps с устройства iPhone или аналогичного смартфона, это значит, что вы уже видели оба способа в действии. Сначала вы видите большой круг, аппроксимирующий ваше текущее местоположение (поиск ближайшей к вам вышки сотовой связи), затем — круг меньшего диаметра (триангуляция с использованием еще одной вышки сотовой связи), затем — точку с уточненными координатами (полученными от спутников GPS).

Причина, по которой я говорю об этом, заключается в том, что, в зависимости от вашего Web-приложения, более высокая точность может вам либо требоваться, либо нет. Возможно, если вам требуется получить список ближайших кинотеатров, режима "низкой точности" будет для вас достаточно. Кинотеатров не так уж и много, даже в густонаселенных городах, и вы в любом случае получите список, в котором кинотеатров будет несколько. С другой стороны, если вы даете кому-либо подробные указания по маршруту следования в режиме реального времени, то вам действительно необходимо знать точные координаты местонахождения пользователя, иначе вы не сможете давать ему указания наподобие "пройдите 20 метров и поверните направо".

Функция `getCurrentPosition()` имеет необязательный третий аргумент — объект `PositionOptions` (**http://www.w3.org/TR/geolocation-API/#position-options**). Для объекта `PositionOptions` можно установить три свойства, которые вкратце перечислены в табл. 6.4. Все эти три свойства необязательны, и вы можете установить любые из них, все или ни одного.

Таблица 6.4. *Свойства объекта* `PositionOptions`

Свойство объекта `PositionOptions`	**Тип**	**Значение по умолчанию**	**Примечание**
`enableHighAccuracy`	`Boolean`	`false`	Значение `true` может замедлить работу
`timeout`	`long`	(отсутствует)	время в миллисекундах
`maximumAge`	`long`	0	время в миллисекундах

Смысл свойства `enableHighAccuracy` в точности соответствует его названию. Если свойство имеет значение `true`, и устройство может его поддерживать, а пользователь согласен предоставлять информацию о своем географическом местоположении, устройство попытается воспользоваться этим свойством. Как iPhone, так и Android имеют отдельные полномочия для позиционирования с высокой и с низкой точностью, поэтому возможно, что вызов функции `getCurrentPosition()` с `enableHighAccuracy:true` может завершиться неудачей, а вызов с `enableHighAccuracy:false` пройдет успешно.

Свойство `timeout` указывает временной интервал в миллисекундах, в течение которого ваше Web-приложение готово ожидать информации о географическом местоположении пользователя. Отсчет времени таймером начинается только после того, как пользователь даст разрешение предпринять попытку вычисления его географических координат. Таймер устанавливается не на действия пользователя, а на работу сети.

Свойство `maximumAge` позволяет устройству отвечать немедленно, отправляя кэшированные координаты. Например, допустим, что вы вызвали функцию `getCurrentPosition()` впервые, пользователь дает согласие на работу с его географическими координатами, и ваша функция обратного вызова вызывается с координатами, вычисленными на момент 10:00 AM. Ровно через минуту, в 10:01 AM, вы снова вызываете функцию `getCurrentPosition()` со свойством `maximumAge`, равным 75 000.

```
navigator.geolocation.getCurrentPosition(
  success_callback, error_callback, {maximumAge: 75000});
```

Этим вы сообщаете, что вам не обязательно нужна *текущая* позиция пользователя. Вас вполне удовлетворит точка, в которой он находился 75 секунд тому назад (75 000 миллисекунд). Устройству известно, где находился пользователь 60 секунд тому назад (60 000 миллисекунд), потому что эти координаты были вычислены, когда вы в первый раз вызвали функцию `getCurrentPosition()`. Таким образом, устройство не утруждает себя вычислением текущей позиции пользователя. Оно просто возвращает ту же самую информацию, что и в первый раз: те же самые широту и долготу, ту же точность с той же самой временной отметкой (10:00 AM).

Прежде чем запрашивать информацию о местонахождении пользователя, вам следует задуматься о том, какая точность вам нужна, и соответствующим образом настроить свойство `enableHighAccuracy`. Если вам требуется определять географические координаты местонахождения пользователя более одного раза, вы можете подумать о том, в течение какого времени информация может оставаться полезной, и соответствующим образом указать значение параметра `maximumAge`. Если вам требуется отслеживать географические координаты местонахождения пользователя *постоянно*, тогда функция `getCurrentPosition()` — не для вас. Вам потребуется другая функция — `watchPosition()`.

Функция `watchPosition()` имеет ту же структуру, что и функция `getCurrentPosition()`. Она принимает две функции обратного вызова, обязательную — на случай успеха и необязательную — для обработки состояний ошибки, а также она может принимать необязательный объект `PositionOptions`, обладающий всеми только что описанными свойствами. Разница заключается в том, что ваша функция обратного вызова будет вызываться *всякий раз, когда изменяются координаты местонахождения пользователя*. Необходимости в активном опросе пользователей об их географических координатах нет. Устройство само определит оптимальный интервал для опроса и будет обращаться к вашей функции обратного

вызова, определив, что пользователь переместился, и его географические координаты изменились. Вы можете использовать этот прием для обновления положения визуального маркера на карте, предоставления пользователю инструкций касательно последующих перемещений, а также для любых других целей, которые вы ставите перед своим приложением. Иными словами, здесь все решаете вы.

Функция `watchPosition()` сама по себе возвращает числовое значение. Возможно, вам захочется где-то эти значения сохранять. Если вы захотите прекратить отслеживание перемещений пользователя, вы можете вызвать метод `clearWatch()` и передать ему это число, после чего устройство прекратит обращаться к вашей функции обратного вызова. Если вы когда-либо пользовались функциями `setInterval()` и `clearInterval()` в JavaScript, то вы уже знаете, как это работает — точно так же.

Как обстоят дела с IE?

Internet Explorer вплоть до выхода версии 9.0 RC1 не поддерживал W3C geolocation API, только что описанный в предыдущем разделе. Но не следует отчаиваться! Существует такой плагин на основе открытого кода, как Gears (**http://tools.google.com/gears/**) от Google, который работает в Windows, Mac OS X, Linux, Windows Mobile и Android. Этот плагин предоставляет более старым браузерам функциональные возможности по поддержке HTML5, аналогичные доступным в современных версиях. Одной из функций, предоставляемых плагином Gears, как раз и является поддержка geolocation API. Это — не совсем то же самое, что W3C geolocation API, но служит той же самой цели.

Говоря о поддержке наследуемых платформ, я должен заметить, что многие старые мобильные телефоны имели собственную поддержку API геолокации, специфичную для своих аппаратных платформ. BlackBerry (**http://www.tonybunce.com/2008/05/08/Blackberry-Browser-Amp-GPS.aspx**), Nokia, Palm **(http://developer.palm.com/index.php?option=com_content&view=article&id=1673#GPS-getCurrentPosition**), OMTP BONDI (**http://bondi.omtp.org/1.0/apis/geolocation.html**) — все они предлагают собственные реализации API геопозиционирования. Разумеется, их принципы работы отличаются от принципа, используемого Gears, который, в свою очередь, отличается от принципа работы W3C geolocation API.

На помощь приходит скрипт geo.js!

Скрипт `geo.js`(**https://code.google.com/p/geo-location-javascript/**) — это библиотека JavaScript на основе открытого кода, поставляемая по лицензии MIT и призванная сгладить различия между W3C geolocation API, Gears API и API, предоставляемыми различными мобильными платформами. Чтобы воспользоваться этой библиотекой, достаточно добавить два элемента `<script>` в конец вашей Web-страницы. С технической точки зрения, их, конечно, можно добавлять куда угодно в пределах вашей страницы, но скрипты в пределах элемента `<head>` замедлят загрузку вашей страницы, так что лучше этого все-таки не делать.

Первый скрипт называется `gears_init.js` (**https://code.google.com/apis/gears/gears_init.js**), и его задача заключается в инициализации Gears, если этот модуль установлен. Второй скрипт называется `geo.js` (**https://geo-location-javascript.googlecode.com/svn/trunk/js/geo.js**). Добавление этих скриптов в код вашей Web-страницы продемонстрировано в листинге 6.6.

Листинг 6.6. Добавление скриптов `gears_init.js` и `geo.js` в код Web-страницы

```
<html>
<head>
  <meta charset="utf-8">
  <title>Dive Into HTML5</title>
</head>
<body>
  ...
  <script src="gears_init.js"></script>
  <script src="geo.js"></script>
</body>
</html>
```

Еще раз — не включайте эти скрипты в состав вашего элемента `<head>`.

Теперь вы сможете пользоваться любым из установленных API геопозиционирования, как показано в листинге 6.7.

Листинг 6.7. Применение API геопозиционирования с помощью скрипта `geo.js`

```
if (geo_position_js.init()) {
  geo_position_js.getCurrentPosition(geo_success, geo_error);
}
```

Давайте действовать пошагово. Сначала вам необходимо вызвать функцию `init()`. Функция `init()` возвращает значение `true`, если доступен поддерживаемый геолокационный API.

```
if (geo_position_js.init()) {
```

Вызов функции `init()` не приводит к фактическому поиску вашего географического местонахождения. Он просто проверяет тот факт, что поиск вашего географического местоположения принципиально возможен. Чтобы проделать это на практике, необходимо вызвать функцию `getCurrentPosition()`:

```
  geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Функция `getCurrentPosition()` сработает, как триггер, инициирующий запрос вашего браузера на ваше разрешение обнаружить ваши географические координаты и предоставить их в общий доступ. Если геопозиционирование осуществляется с помощью Gears, то на экране появится диалоговое окно, запрашивающее вашего разрешения использовать Gears через Web. Если ваш браузер обеспечивает встроенную

поддержку geolocation API, это диалоговое окно будет выглядеть по-другому. Например, Firefox 3.5 обеспечивает встроенную поддержку geolocation API. Если вы попытаетесь определить свое местонахождение в Firefox 3.5, на экране в верхней части страницы появится информационная панель (infobar), запрашивающая, желаете ли вы предоставить данные о своем географическом местоположении данному Web-сайту.

Функция `getCurrentPosition()` принимает в качестве аргументов две функции обратного вызова. Если функция `getCurrentPosition()` успешно определит географические координаты точки вашего местоположения — иными словами, если вы дадите ей свою санкцию, а geolocation API действительно выполнит свою работу так, как ожидалось, то этот интерфейс вызовет функцию, переданную ему в качестве первого аргумента. В рассматриваемом примере функция обратного вызова, к которой происходит обращение в случае успеха, называется `geo_success`.

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Функция обратного вызова, к которой осуществляется обращение в случае успеха, принимает единственный аргумент, который содержит геолокационную информацию.

Функция обратного вызова в случае успеха выглядит так, как показано в листинге 6.8.

Листинг 6.8. Функция geo_success в случае успеха

```
function geo_success(p) {
  alert("Found you at latitude " + p.coords.latitude +
        ", longitude " + p.coords.longitude);
}
```

Если функции `getCurrentPosition()` не удалось определить ваши координаты — либо потому, что вы отклонили запрос на предоставление информации, либо потому, что в работе geolocation API по той или иной причине произошел сбой, то будет вызвана функция обратного вызова, переданная в качестве второго аргумента. В рассматриваемом примере функция называется `geo_error`:

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Функция обратного вызова, к которой происходит обращение в случае неудачи, не имеет аргументов.

Эта функция выглядит так, как показано в листинге 6.9.

Листинг 6.9. Функция обратного вызова в случае неудачи

```
function geo_error() {
  alert("Could not find you!");
}
```

В настоящее время `geo.js` не поддерживает функции `watchPosition()`. Если вам требуется отслеживать позицию пользователя в режиме реального времени, вам нужно будет самостоятельно вести активный опрос `getCurrentPosition()`.

Полноценный "живой" пример

Рассмотрим полнофункциональный "живой" пример использования `geo.js` с тем, чтобы попытаться определить свои текущие географические координаты и отобразить ваше текущее местоположение на карте (**http://diveintohtml5.org/geolocation.html**).

Как это работает? Давайте посмотрим. При загрузке страницы она вызовет функцию `geo_position_js.init()`, чтобы определить, доступна ли вам возможность геопозиционирования через один из интерфейсов, поддерживаемых `geo.js`. Если это так, функция создает ссылку, по которой вы можете проследовать с тем, чтобы определить свои текущие географические координаты. Когда вы пройдете по этой ссылке, будет вызвана функция `lookup_location()`, показанная в листинге 6.10.

Листинг 6.10. Функция `lookup_location()`

```
function lookup_location() {
  geo_position_js.getCurrentPosition(show_map, show_map_error);
}
```

Если вы даете свое согласие на отслеживание вашего текущего географического местоположения, и серверная часть геолокационного сервиса действительно способна определить ваше местонахождение, то `geo.js` вызовет первую функцию обратного вызова, `show_map()`, с единственным аргументом, `loc`. Объект `loc` имеет свойство `coords`, которое указывает широту, долготу и точность. (Рассматриваемый пример информации о точности не включает.) Остальной код функции `show_map()` использует API Google Maps для настройки встроенной карты, как показано в листинге 6.11.

Листинг 6.11. Код функции `show_map()`

```
function show_map(loc) {
  $("#geo-wrapper").css({'width':'320px','height':'350px'});
  var map = new GMap2(document.getElementById("geo-wrapper"));
  var center = new GLatLng(loc.coords.latitude, loc.coords.longitude);
  map.setCenter(center, 14);
  map.addControl(new GSmallMapControl());
  map.addControl(new GMapTypeControl());
   map.addOverlay(new GMarker(center, {draggable: false, title: "You are here
(more or less)"}));
}
```

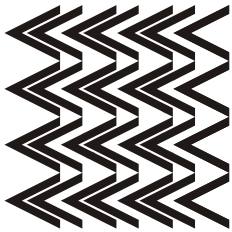
Если `geo.js` не может определить ваши координаты, вызывается вторая функция обратного вызова, `show_map_error()`, код которой представлен в листинге 6.12.

Листинг 6.12. Код функции `show_map_error()`

```
function show_map_error() {
   $("#live-geolocation").html('Unable to determine your location.');
}
```

Материалы, рекомендуемые для дальнейшего чтения

- Официальная документация по W3C geolocation API — **http://www.w3.org/TR/geolocation-API/**
- Официальная документация по Google Gears — **http://tools.google.com/gears/**
- BlackBerry geolocation API — **http://www.tonybunce.com/2008/05/08/Blackberry-Browser-Amp-GPS.aspx**
- Nokia geolocation API — **http://wiki.forum.nokia.com/index.php/Bondi_Widget_porting_example_-_geolocation_API**
- Palm geolocation API — **http://developer.palm.com/index.php?option=com_content&view=article&id=1673#GPS-getCurrentPosition**
- OMTP BONDI geolocation API — **http://bondi.omtp.org/1.0/apis/geolocation.html**
- `geo.js`, скрипт-"обертка" для геолокационного API — **https://code.google.com/p/geo-location-javascript/**
- Описание geolocation API в Internet Explorer 9 — **http://msdn.microsoft.com/en-us/ie/ff468705.aspx#_HTML5_geolocation**



Глава 7

Прошлое, настоящее и будущее Web-приложений для хранения данных

Постоянное локальное хранение информации — это та область, где полноценные клиентские приложения, обладающие встроенной поддержкой локального хранения информации, традиционно имели преимущество перед Web-приложениями. Для полноценных клиентских приложений операционная система, как правило, предоставляет уровень абстракций для хранения и извлечения таких специфичных для приложения данных, как настройки приложения и статус времени выполнения. Эти значения могут храниться в системном реестре (только для Windows), INI-файлах, XML-файлах или других компонентах системы, установленных для конкретной платформы. Если ваш клиент нуждается в локальном хранении других данных, а не только пар "ключ/значение", вы можете встроить в него собственную базу данных, разработать собственный формат хранения данных или выработать целый ряд других решений.

Исторически сложилось так, что Web-приложения были лишены этих удобных возможностей. Cookie-файлы были изобретены на ранних этапах развития Web, и, естественно, их можно использовать для локального хранения небольших объемов данных. Но они имеют три крупных недостатка:

- cookie-файлы включаются в каждый запрос HTTP и, таким образом, замедляют работу вашего Web-приложения бесполезной пересылкой данных туда и обратно;
- cookie-файлы включаются в каждый запрос HTTP, поэтому данные постоянно пересылаются через Интернет в незашифрованном виде (за исключением тех случаев, когда все ваше приложение работает через SSL);
- cookie-файлы имеют ограничение по размеру (до 4 Кбайт для каждого файла). Этого уже достаточно, чтобы существенно замедлить работу вашего приложения, но явно мало для того, чтобы предоставлять реальную пользу.

А теперь давайте посмотрим, что нам требуется в реальности:

- большой объем пространства, чтобы клиентское приложение могло хранить свои данные, которые должны сохраняться между обновлениями страницы;
- возможность сохранения данных без необходимости их передачи на сервер.

До появления HTML5, все попытки добиться этих целей по различным причинам оказывались неудовлетворительными.

Краткая история локального хранения данных до появления HTML5

Изначально в распоряжении пользователей был только один браузер — Internet Explorer. Ну, по крайней мере, ребята из Microsoft хотели, чтобы все остальные думали именно так. На тот момент, в рамках Великих браузерных войн (**https://secure.wikimedia.org/wikipedia/en/wiki/Browser_wars#The_first_browser_war**), корпорация Microsoft ввела множество новшеств и включила их в свой продукт Internet Explorer, стремясь прикончить все остальные конкурирующие браузеры. Одно из этих новшеств называлось "Элементы поведения DHTML или SHTML Behaviors" (**http://msdn.microsoft.com/en-us/library/ms531078%28v=VS.85%29.aspx**), а одним из этих "стереотипов поведения" назывался `userData` (**http://msdn.microsoft.com/en-us/library/ms531424%28VS.85%29.aspx**).

`userData` позволяет Web-страницам хранить 64 Кбайт данных на домен, в виде иерархической структуры на основе XML. Доверенные домены, например, интранет-сайты, могут хранить в 10 раз больший объем информации. Ну и вспомните знаменитое высказывание Билла Гейтса о том, что "640 Кбайт должно хватать каждому", хотя сам он и отрицал тот факт, что он такие вещи говорил (**https://secure.wikimedia.org/wikiquote/en/wiki/Bill_Gates#Misattributed**). IE не предоставляет ни в какой форме диалогового окна с запросом полномочий, как и не допускает никаких разрешений на увеличение объема пространства для хранения таких данных.

В 2002 году корпорация Adobe ввела в своем продукте Flash 6 функцию, которая получила крайне неудачное название "Flash cookies". В среде Flash эта функция известна под более корректным наименованием "локальных разделяемых объектов" (Local Shared Objects)[1]. Вкратце говоря, это позволяет объектам Flash хранить до 100 Кбайт данных на домен. Брэд Нойберг (Brad Neuberg) разработал ранний прототип "моста" от Flash к JavaScript, называющийся AMASS (AJAX Massive Storage System)[2], но он был ограничен некоторыми особенностями дизайна Flash. В 2006 году, с появлением в продукте Flash 8 интерфейса `ExternalInterface` (**http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/external/ExternalInterface.html**), доступ к LSO из JavaScript сделался на порядок быстрее и проще. Брэд переписал AMASS и интегрировал его в популярный набор инструментов Dojo Toolkit (**http://dojotoolkit.org/**) под ярлыком `dojox.storage` (**http://dojotoolkit.org/api/dojox.storage.manager**). Flash "бесплатно" предоставляет по 100 Кбайт пространства для хранения данных на каждый домен. Помимо этого, Flash запрашивает пользователей, не желают ли они увеличить объем хранимых данных на порядок (1 Мбайт, 10 Мбайт и так далее) каждый раз, когда объем этих данных приближается к установленной на данный момент верхней границе.

---

[1] См. **https://secure.wikimedia.org/wikipedia/en/wiki/Local_Shared_Object**, **http://www.woweb.ru/publ/3-1-0-259**, **http://www.adobe.com/products/flashplayer/articles/lso/**.

[2] См. **http://codinginparadise.org/weblog/2005/10/amass-ajax-massive-storage-system.html**.

В 2007 году корпорация Google запустила Gears (**http://gears.google.com/**), дополнительный модуль для браузера (плагин) на основе открытого кода, цель которого заключалась в предоставлении браузерам дополнительных возможностей. Ранее мы уже обсуждали Gears в контексте предоставления возможностей API геопозиционирования в версиях Internet Explorer, более ранних, чем IE 9. Gears предоставляет API к встроенной базе данных SQL (**https://code.google.com/apis/gears/api_database.html**) на основе SQLite (**http://www.sqlite.org/**). После того как разрешение от пользователя будет получено (достаточно сделать это один раз), Gears сможет хранить любое количество данных на домен в таблицах базы данных SQL.

В течение некоторого времени Брэд Нойберг и другие разработчики продолжали работать над `dojox.storage` с тем, чтобы предоставить унифицированный интерфейс к множеству различных плагинов и API. К 2009 году `dojox.storage` уже мог автоматически определять (и предоставлять унифицированный интерфейс, работающий "поверх") для таких плагинов, как Adobe Flash, Gears, Adobe AIR и ранних прототипов хранилища данных HTML5, которое было реализовано только в более старых версиях Firefox.

При внимательном изучении всех этих решений становится очевидным один общий подход: все решения либо являются специфичными для одного конкретного браузера, либо полагаются на сторонний плагин. Несмотря на героические усилия документировать все различия (в `dojox.storage`), все они имели настолько радикально различающиеся интерфейсы, такие разные ограничения по доступному пространству для хранения данных и такие различные пользовательские интерфейсы, что все эти труды не увенчались успехом. Именно эту проблему и призвана решить спецификация HTML5: обеспечить стандартный API, реализованный на встроенном уровне и непротиворечивый, единообразный для всех браузеров, исключающий необходимость полагаться на сторонние плагины.

Введение в хранилище данных HTML5

То, что я называю "хранилищем данных HTML5 Storage", носит официальное название "Web-хранилища" или Web Storage (**http://dev.w3.org/html5/webstorage/**) и когда-то действительно являлось полноправной частью спецификации HTML5, но впоследствии, по политическим мотивам (не представляющим для нас интереса) было выделено в отдельную спецификацию. Некоторые поставщики браузеров называют его также "Local Storage" или "DOM Storage". Система именования становится еще более сложной и запутанной, когда речь заходит о некоторых близких стандартах с похожими именами, которые я буду обсуждать далее в этой главе.

Итак, что же представляет собой стандарт HTML5 Storage? Проще говоря, это — способ, к которому Web-страницы могут прибегнуть для организации локального хранения пар "ключ/значение", которые будут использоваться Web-браузерами, с которыми работает клиент. Как и cookie-файлы, эти данные сохраняются, даже если вы уходите с Web-сайта, закрываете вкладку в браузере или окно, в котором просматривается данная страница, или же совершаете другие подобные действия. В отличие от cookie-файлов, эти данные никогда не передаются

на удаленный Web-сервер (за исключением тех случаев, когда вы сами отправляете их туда вручную). В отличие от всех предшествующих попыток обеспечить постоянное локальное хранение данных, эта технология реализована в Web-браузерах на встроенном уровне, так что она будет доступна даже в том случае, если у вас нет доступа к сторонним плагинам для Web-браузеров.

Каких браузеров это касается? Как минимум, к их числу относятся новейшие версии всех современных браузеров, где декларируется поддержка HTML5 Storage, включая даже Internet Explorer! Сведения о поддержке HTML5 Storage в современных версиях браузеров и мобильных устройствах приведены в табл. 7.1.

***Таблица 7.1.** Поддержка технологии HTML5 в современных браузерах и мобильных устройствах*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
8.0+	3.5+	4.0+	4.0+	10.5+	2.0+	2.0+

Из вашего кода на JavaScript вы можете получать доступ к HTML5 Storage через объект `localStorage` в составе глобального объекта `window`. Прежде чем пользоваться локальным хранилищем HTML5, вам необходимо предварительно убедиться в том, что оно поддерживается вашим браузером. Делается это так, как показано в листинге 7.1.

Листинг 7.1. Проверка наличия поддержки HTML5 Storage в вашем браузере

```
function supports_html5_storage() {
  try {
    return 'localStorage' in window && window['localStorage'] !== null;
  } catch (e) {
    return false;
  }
}
```

Вместо самостоятельного написания функции `supports_html5_storage()` вы можете использовать библиотеку `Modernizr`, чтобы выявить наличие поддержки HTML5 Storage, как показано в листинге 7.2.

Листинг 7.2. Проверка наличия поддержки HTML5 Storage с помощью библиотеки Modernizr

```
if (Modernizr.localstorage) {
  // доступен объект window.localStorage!
} else {
  // нет встроенной поддержки для HTML5 storage :(
  // попробуйте воспользоваться dojox.storage
..// или другим сторонним решением
```

Использование хранилища HTML5

Хранилище данных HTML5 (HTML5 Storage) основывается на именованных парах "ключ/значение". Вы сохраняете данные в именованном ключе, а затем вы извлекаете эти данные по имени того же самого ключа. Именованный ключ представляет собой строку. Данные могут иметь любой тип, поддерживаемый JavaScript, включая строки, логические значения типа `Boolean`, целые числа, числа с плавающей точкой (значения типа `float`). Однако фактически данные хранятся как строки. Если вы сохраняете или извлекаете что-то, отличное от строк, вам понадобится использовать функции наподобие `parseInt()` или `parseFloat()`, чтобы принудительным образом установить для ваших извлеченных данных ожидаемый тип JavaScript, как показано в листинге 7.3.

Листинг 7.3. При сохранении или извлечении данных, отличных от строк, необходимо принудительно устанавливать тип данных, ожидаемый JavaScript

```
interface Storage {
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key, in any data);
};
```

Вызов `setItem()` с именованным ключом, который уже существует, приведет к затиранию предыдущего значения без предупреждения. Вызов функции `getItem()` с указанием несуществующего ключа вернет `null` вместо того, чтобы "выбросить" исключение.

Как и с остальными объектами JavaScript, вы можете обращаться с объектом `localStorage` как с ассоциативным массивом. Вместо использования методов `getItem()` и `setItem()`, вы можете просто использовать квадратные скобки. Рассмотрим, например, фрагмент кода, приведенный в листинге 7.4.

Листинг 7.4. Работа с объектом `localStorage` с помощью методов `getItem()` и `setItem()`

```
var foo = localStorage.getItem("bar");
// ...
localStorage.setItem("bar", foo);
```

Его можно переписать так, чтобы использовать синтаксис с применением квадратных скобок, как показано в листинге 7.5.

Листинг 7.5. Работа с объектом `localStorage` с помощью синтаксиса с применением квадратных скобок

```
var foo = localStorage["bar"];
// ...
localStorage["bar"] = foo;
```

Кроме того, существуют методы удаления значения для данного именованного ключа, и очистки всего пространства хранения (т. е. одновременного удаления всех ключей и значений), применение которых продемонстрировано в листинге 7.6.

Листинг 7.6. Методы удаления значения для данного именованного ключа и очистки всего пространства хранения

```
interface Storage {
  deleter void removeItem(in DOMString key);
  void clear();
};
```

Вызов метода `removeItem()` с несуществующим ключом не приведет ни к какому действию.

Наконец, существует свойство, позволяющее получить общее количество значений в области хранения, а также итерациями перебирать все их значения по индексу (чтобы получить имя каждого ключа), применение которого продемонстрировано в листинге 7.7.

Листинг 7.7. Использование свойства, позволяющего получить общее количество значений

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
};
```

Если вы вызовете метод `key()` с индексом, который не находится в интервале от 0 до (`length-1`), то функция возвратит `null`.

Отслеживание изменений в области хранения данных HTML5

Если вы хотите программно отслеживать изменения области хранения данных, вы можете захватывать события, связанные с хранилищем данных. Событие, связанное с хранилищем данных, срабатывает по отношению к объекту `window` всякий раз, когда происходят вызовы `setItem()`, `removeItem()` или `clear()`, которые *реально производят некоторые изменения*. Например, если вы установите для элемента существующее значение или вызовете метод `clear()` при отсутствии именованных ключей, то событие, связанное с хранилищем данных, не будет активировано, потому что на самом деле в области хранения данных ничего не произойдет.

Событие, связанное с хранением данных, поддерживается везде, где поддерживается объект `localStorage`, в том числе — и в Internet Explorer 8. IE 8 не поддерживает `addEventListener` по стандарту W3C (хотя такая поддержка будет добавлена в IE 9). Следовательно, чтобы программно обрабатывать события, связанные

с хранилищем данных, вам необходимо проверить, какой механизм обработки событий поддерживает браузер. Такая проверка продемонстрирована в листинге 7.8. Если вы уже делали это раньше для других событий, вы можете пропустить весь излагаемый материал до конца этого раздела. Перехват событий, связанных с хранилищем данных, работает точно так же, как и перехват других событий. Если вы предпочитаете использовать jQuery или какую-нибудь другую библиотеку JavaScript для регистрации ваших обработчиков событий, вы можете делать и это тоже.

Листинг 7.8. Проверка механизма обработки событий, поддерживаемого браузером

```
if (window.addEventListener) {
  window.addEventListener("storage", handle_storage, false);
} else {
  window.attachEvent("onstorage", handle_storage);
};
```

Функция обратного вызова `handle_storage` может вызываться с использованием объекта `StorageEvent`, за исключением ситуации, когда вы пользуетесь Internet Explorer, где объект события хранится в событии window. Вызов функции `handle_storage` продемонстрирован в листинге 7.9.

Листинг 7.9. Вызов функции `handle_storage`

```
function handle_storage(e) {
  if (!e) { e = window.event; }
}
```

На данном этапе, переменная `e` будет объектом `StorageEvent`, который имеет полезные свойства, перечисленные в табл. 7.2.

***Таблица 7.2.** Свойства объекта `StorageEvent`*

Свойство	Тип	Описание
`key`	строка	Именованный ключ, который был добавлен, удален или модифицирован
`oldValue`	любой	Предыдущее значение (ныне перезаписанное) или `null`, если был добавлен новый элемент
`newValue`	любой	Новое значение или null, если элемент был удален
`url*`	строка	Страница, которая вызвала метод, инициировавший изменение

* Примечание: Свойство `url` изначально называлось `uri`. Некоторые браузеры поставлялись с этим свойством еще до изменения спецификации. Для максимальной совместимости, вам необходимо убедиться в существовании свойства `url` и, если это не так, проверить вместо этого существование свойства `uri`.

Событие, связанное с хранением данных, отменить нельзя. Из обработчика функции обратного вызова `handle_storage` нет способа остановить внесение изменения. Это — просто способ для браузера сообщить вам о происходящем, как и о том, что вы ничего больше не можете изменить.

Ограничения в текущих версиях браузеров

Говоря об истории развития "хаков", связанных с локальным хранилищем, с помощью сторонних плагинов, я особо отметил ограничения каждого из методов, включая ограничения по пространству, доступному для хранения. Но я только сейчас вспомнил, что забыл рассказать об ограничениях, которые на сегодняшний момент имеет стандартизованное хранилище HTML5. Сначала я просто дам вам все ответы, а затем и объясню их. Ответы, в порядке их значимости, будут такими: "5 Мбайт", "`QUOTA_EXCEEDED_ERR`" и "нет".

"5 Мбайт" — это тот объем пространства, который по умолчанию отводится каждому "источнику" (origin)[1]. Это ограничение на удивление последовательно и непротиворечиво для всех браузеров, хотя оно сформулировано просто как пожелание к спецификации хранилища HTML5. Факт, который следует иметь в виду, заключается в том, что вы храните строки, а не данные в исходном формате. Если вы храните большое количество целых чисел или чисел с плавающей точкой (`float`), то разница в представлении может привести к существенной добавке. Каждая цифра в числе с плавающей точкой хранится как символ, а не как традиционное число с плавающей точкой.

"`QUOTA_EXCEEDED_ERR`" — это исключение, которое будет инициировано в случае превышения квоты на хранение объемом 5 Мбайт.

"Нет" — это ответ на следующий очевидный вопрос, "можно ли запросить у пользователя больший объем пространства"? На момент написания этих строк, ни один из браузеров не поддерживает никакого механизма, воспользовавшись которым Web-разработчики могли бы запросить дополнительные объемы свободного пространства. Некоторые браузеры (такие, как Opera[2]) позволяют пользователям управлять квотами на хранение данных для каждого сайта, но это — всего лишь действие, инициируемое пользователем, а не нечто такое, что вы, как Web-разработчик, могли бы встроить в свое Web-приложение.

Хранилище HTML5 в действии

Давайте рассмотрим, как работает хранилище данных HTML5. Вспомните игру Halma, которую мы написали в *главе 5*, посвященной элементу `<canvas>`. С этой игрой связана небольшая проблема: если вы закроете окно браузера, не закончив игру, то вы потеряете все данные об этой игре. Но с помощью HTML5 Storage мы можем сохранить ход игры локально, с помощью самого браузера. Вот пример вкладки браузера, которую вы затем можете открыть повторно и чтобы просмотреть

[1] См. **http://www.whatwg.org/specs/web-apps/current-work/multipage/origin-0.html#origin-0**.
[2] См. **http://dev.opera.com/articles/view/web-storage/**.

"живую демонстрацию" (**http://diveintohtml5.org/examples/localstorage-halma.html**). Выполните несколько ходов, закройте вкладку, а затем снова ее откройте. Если ваш браузер поддерживает HTML5 Storage, то демонстрационная страница "магическим образом" запомнит позицию, которую вы создали на тот момент, когда покинули игру, включая количество выполненных вами ходов, позицию каждой фишки на доске и даже то, была или не была выбрана конкретная фишка.

Как это работает? Каждый раз, когда в ходе игры происходит изменение, мы вызываем функцию `saveGameState()`, код которой показан в листинге 7.10.

Листинг 7.10. Код функции `saveGameState()`, которая осуществляет локальное сохранение текущего статуса игры

```
function saveGameState() {
    if (!supportsLocalStorage()) { return false; }
    localStorage["halma.game.in.progress"] = gGameInProgress;
    for (var i = 0; i < kNumPieces; i++) {
localStorage["halma.piece." + i + ".row"] = gPieces[i].row;
        localStorage["halma.piece." + i + ".column"] = gPieces[i].column;
    }
    localStorage["halma.selectedpiece"] = gSelectedPieceIndex;
    localStorage["halma.selectedpiecehasmoved"] = gSelectedPieceHasMoved;
    localStorage["halma.movecount"] = gMoveCount;
    return true;
}
```

Как видите, функция создает объект `localStorage`, позволяющий указать, продолжается ли еще игра (`gGameInProgress`, тип `Boolean`). Если это так, то игра проходит итерации (`gPieces`, массив JavaScript) и сохраняет ряд и строку для каждой фишки. Затем программа сохраняет дополнительный статус игры, в том числе, с указанием, выбрана ли фишка для хода (`gSelectedPieceIndex`, целое число), находится ли фишка в середине (потенциально длинной) последовательности шагов (`gSelectedPieceHasMoved`, значение типа `Boolean`) и включая общее количество шагов, выполненных до сих пор (`gMoveCount`, значение типа `integer`).

При автоматической загрузке страницы, вместо автоматического вызова функции `newGame()`, которая переустановила бы эти значения на жестко закодированные, вы вызовете функцию `resumeGame()`. С использованием HTML5 Storage, функция `resumeGame()` проверит, не был ли локально сохранен текущий статус игры. Если это окажется так, то функция продолжит выполнение, восстановив значения с помощью объекта `localStorage`, как показано в листинге 7.11.

Листинг 7.11. Функция `resumeGame()` проверяет, не был ли локально сохранен статус игры и возобновляет начатую игру, если это так

```
function resumeGame() {
    if (!supportsLocalStorage()) { return false; }
    gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
```

```
    if (!gGameInProgress) { return false; }
    gPieces = new Array(kNumPieces);
    for (var i = 0; i < kNumPieces; i++) {
       var row = parseInt(localStorage["halma.piece." + i + ".row"]);
       var column = parseInt(localStorage["halma.piece." + i + ".column"]);
       gPieces[i] = new Cell(row, column);
    }
    gNumPieces = kNumPieces;
    gSelectedPieceIndex = parseInt(localStorage["halma.selectedpiece"]);
    gSelectedPieceHasMoved = localStorage["halma.selectedpiecehasmoved"] == "true";
    gMoveCount = parseInt(localStorage["halma.movecount"]);
    drawBoard();
    return true;
}
```

Наиболее важным аспектом этой функции является та "ловушка", которую я упомянул ранее в этой главе и о которой сейчас скажу подробнее. Данные хранятся как строки. Если вы храните не строки, а что-то другое, вам нужно будет принимать самостоятельные усилия по преобразованию этих данных при их извлечении. Например, флаг, указывающий, завершена ли игра (`gGameInProgress`) — это логическое значение (`Boolean`). В функции `saveGameState()` мы просто сохранили его, не заботясь о типе данных этого значения:

```
localStorage["halma.game.in.progress"] = gGameInProgress;
```

Но в функции `resumeGame()` нам необходимо обращаться со значением, извлеченным из локального хранилища, как со строкой, и вручную воссоздать правильное логическое значение (тип `Boolean`):

```
gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
```

Аналогично, количество выполненных ходов хранится в переменной `gMoveCount` как целое число. В функции `saveGameState()` мы его просто сохранили:

```
localStorage["halma.movecount"] = gMoveCount;
```

Но в функции `resumeGame()` мы должны преобразовать это значение в целое число, используя функцию `parseInt()`, встроенную в JavaScript:

```
gMoveCount = parseInt(localStorage["halma.movecount"]);
```

За пределами именованных пар "ключ-значение": Конкурирующие воззрения

Хотя все прошлое буквально нашпиговано "хаками" и обходными методами, текущее состояние хранилища HTML5 на удивление напоминает "путь, устланный розами". Новый API уже стандартизован и реализован в большинстве браузеров, на всех основных платформах и устройствах. Как Web-разработчик, вы вряд ли видите такие ситуации ежедневно, не так ли? Но для реальной жизни локальное хранилище — это не просто "5 Мбайт именованных пар ключей и значений", и будущее

постоянного локального хранилища данных выглядит… как бы это сказать помягче… ну ладно, скажем, что существуют и другие, конкурирующие взгляды на его будущее.

Один из этих взглядов — это акроним, который вам, вероятно, уже давно и хорошо известен: SQL. В 2007 году корпорация Google запустила Gears (**http://gears.google.com/**), кросс-браузерный плагин на основе открытого кода, который включал встроенную базу данных на основе SQLite (**http://www.sqlite.org/**). Этот ранний прототип впоследствии повлиял на создание спецификации Web SQL Database (**http://dev.w3.org/html5/webdatabase/**). Web SQL Database (ранее известная "WebDB") предоставляет тонкую "обертку" вокруг базы данных SQL, позволяя вам выполнять с помощью JavaScript различные задачи. Так, код, приведенный в листинге 7.12, работает в 4 браузерах.

Листинг 7.12. Использование Web SQL Database

```
openDatabase('documents', '1.0', 'Local document storage', 5*1024*1024, func-
tion (db) {
  db.changeVersion('', '1.0', function (t) {
    t.executeSql('CREATE TABLE docids (id, name)');
  }, error);
});
```

Как вы можете видеть, большинство событий происходит в строке, которую вы передаете методу `executeSql`. Эта строка может представлять собой любое поддерживаемое утверждение SQL, включая `SELECT`, `UPDATE`, `INSERT` и `DELETE`. Все происходит точно так же, как при программировании серверной части базы данных, за исключением того, что все операции вы выполняете из JavaScript! Вот это да!

Спецификация Web SQL Database реализована на четырех браузерах и платформах, перечисленных в табл. 7.3.

Таблица 7.3. *Поддержка Web SQL Database в современных браузерах и мобильных устройствах*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
·	·	4.0+	4.0+	10.5+	3.0+	2.0+

Конечно, если за свою жизнь вы успели поработать с несколькими программными продуктами, предназначенными для обработки баз данных, вы наверняка знаете, что SQL — это не просто маркетинговый термин и не просто стандарт. (Некоторые скажут то же самое и про HTML5, но мы пока не будем углубляться в эту тему.) Естественно, существует и "чистая" спецификация SQL, которая называется SQL-92 (**http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt**), но во всем мире не существует такого сервера баз данных, который соответствовал бы только этой спецификации, и никакой другой. Например, достаточно сказать, что в мире

существуют Oracle SQL, Microsoft SQL, MySQL SQL, PostgreSQL SQL, SQLite SQL — и все они нашли свою "рыночную нишу". Само собой разумеется, что каждый из перечисленных продуктов со временем добавляет в SQL свои собственные индивидуальные черты, поэтому даже сказать "SQLite SQL" — еще не достаточно для того, чтобы объяснить, что именно вы имели в виду. Чтобы добиться поставленной цели, вам необходимо сказать, например, "версия SQL, которая поставлялась вместе с SQLite версии X.Y.Z".

Все до сих пор сказанное наводит на мысль о необходимости сделать следующее заявление, на основе действующей на текущий момент спецификации Web SQL Database (**http://dvcs.w3.org/hg/IndexedDB/raw-file/tip/Overview.html**):

> "Данная спецификация достигла вершины своего развития: все заинтересованные стороны воспользовались одной и той же серверной частью SQL (SQLite), но нам (сообществу) требуется множество независимых реализаций, чтобы продолжить работу по разработке стандартов. До тех пор, пока еще один заинтересованный разработчик не проявит интереса к данной теме, это описание диалекта SQL останется просто ссылкой на SQLite, а для стандарта это неприемлемо".

После того как я сделал это заявление, я хотел бы представить вам еще один конкурирующий взгляд на современные Web-приложения с постоянным локальным хранилищем данных, именно: Индексированный API баз данных (Indexed Database API), ранее известный как "WebSimpleDB", а теперь получивший распространение под названием "IndexedDB".

Indexed Database API (**http://dvcs.w3.org/hg/IndexedDB/raw-file/tip/Overview.html**) предоставляет так называемое хранилище объектов (object store). Хранилище объектов использует многие концепции, присущие и базе данных SQL. В нем существуют "базы данных", в которых содержатся "записи", и каждая запись состоит из набора нескольких "полей". Каждое поле имеет конкретный тип данных, который определяется при создании "базы данных". Вы можете выбрать подмножество записей, а затем выполнить их энумерацию (перечисление) с помощью "курсора". Изменения в хранилище объектов обрабатываются в пределах "транзакций".

Если вы имеете некоторый опыт в области программирования баз данных SQL, то все упомянутые термины должны быть вам знакомы. Основное отличие здесь состоит в том, что хранилище объектов не имеет структурного языка запросов (structured query language). Поэтому, работая с ним, вы не будете строить утверждения наподобие `SELECT * from USERS where ACTIVE = 'Y'`. Вместо этого вы будете пользоваться методами, предоставляемыми хранилищем объектов, чтобы открыть курсор над базой данных `USERS`, выполнить перечисление записей, отфильтровать записи для неактивных пользователей и использовать аксессорные методы (accessor methods) для получения значений каждого поля оставшихся записей. Краткое описание IndexedDB (**https://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/**) послужит хорошим руководством, которое поможет вам понять, как работает IndexedDB, предоставляя пошаговые сравнения между IndexedDB и Web SQL Database.

На момент написания этих строк, IndexedDB была реализована только в бета-версии Firefox 4 (**https://developer.mozilla.org/en/Firefox_4_for_developers**). По контрасту, разработчики Mozilla объявили, что они не планируют реализовать Web SQL Database (**https://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/**). С другой стороны, корпорация Google заявила, что они рассматривают возможность поддержки IndexedDB для Chromium и Google Chrome (**http://www.chromium.org/developers/design-documents/indexeddb**). Даже корпорация Microsoft заявила, что IndexedDB — "это замечательное решение для Web" (**http://blogs.msdn.com/b/ie/archive/2010/03/09/working-with-the-html5-community.aspx**).

Так что же вы, как Web-разработчик, можете делать с IndexedDB? На данный момент, практически ничего, исключая нескольких демонстрационных примеров. А через некоторое время, скажем, через год? Может быть, тоже ничего. По крайней мере, ознакомьтесь с источниками, рекомендованными для дополнительного чтения — среди них есть хорошие учебные руководства, которые помогут вам сделать первые шаги в изучении этой важной и интересной темы.

Материалы для дополнительного изучения

Хранилище HTML5:

- Спецификация хранилища данных HTML5 — **http://dev.w3.org/html5/webstorage/**
- Введение в хранилище DOM (документация MSDN) — **http://msdn.microsoft.com/en-us/library/cc197062%28VS.85%29.aspx**
- *"Web Storage: easier, more powerful client-side data storage"* (документация в сообществе разработчиков Opera) — **http://dev.opera.com/articles/view/web-storage/**
- *"DOM Storage"* (**https://developer.mozilla.org/en/dom/storage**) — документация Центра разработчиков Mozilla (Mozilla Developer Center). Примечание: большая часть этой страницы посвящена реализации прототипа объекта `globalStorage` в Firefox, представляющей собой нестандартное введение в `localStorage`. Mozilla добавляет поддержку для стандартного интерфейса localStorage в Firefox 3.5.
- "Unlock local storage for mobile Web applications with HTML5", учебный курс IBM DeveloperWorks — **http://www.ibm.com/developerworks/xml/library/x-html5mobile2/**

Ранние работы Брэда Нойберга (Brad Neuberg) и соавторов (до HTML5):

- "Internet Explorer Has Native Support for Persistence?" (об объекте `userData` в IE) — **http://codinginparadise.org/weblog/2005/08/ajax-internet-explorer-has-native.html**
- "Dojo Storage", часть более крупного руководства по (на данный момент устаревшей) библиотеке Dojo Offline — **https://docs.google.com/View?docid=dhkhksk4_8gdp9gr#dojo_storage**
- Справочное руководство по `dojox.storage` — **http://dojotoolkit.org/api/dojox.storage.manager**
- Репозиторий подверсий `dojox.storage` — **http://svn.dojotoolkit.org/src/dojox/trunk/storage/**

Web SQL Database:

- Спецификация Web SQL Database — **http://dev.w3.org/html5/webdatabase/**
- Введение в базы данных Web SQL — **http://html5doctor.com/introducing-web-sql-databases/**
- Демонстрация Web Database — **http://html5demos.com/database**

Скрипт `persistence.js`, построенный на базе Web SQL Database и Gears — **http://zef.me/2774/persistence-js-an-asynchronous-javascript-orm-for-html5gears**

IndexedDB:

- Спецификация Indexed Database API — **http://dvcs.w3.org/hg/IndexedDB/raw-file/tip/Overview.html**
- *"Beyond HTML5: Database APIs and Road to IndexedDB"* (документация Центра разработки Mozilla) — **https://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/**
- *"Firefox 4: An Early walk-thfrough of IndexedDB"* (документация Центра разработки Mozilla) — **https://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/**

Глава 8

Давайте возьмем все это в автономный режим

Что представляет собой автономное Web-приложение? На первый взгляд, кажется, что это — какое-то терминологическое недоразумение. Во всяком случае, противоречие кажется очевидным. Web-страницы — это то, что мы скачиваем и визуализируем. Скачивание подразумевает наличие сетевого соединения. Как мы можем что-то скачать, если у нас на данный момент нет соединения с Интернетом? Естественно, сделать этого мы не можем. Но при наличии соединения возможность скачивания появляется. Именно по этому принципу и работают автономные приложения HTML5.

В простейшей своей форме автономное Web-приложение представляет собой список URL — файлов HTML, CSS, JavaScript, изображений и ресурсов любого другого типа. Домашняя страница автономного Web-приложения указывает на этот список, который называется файлом манифеста (manifest file), который представляет собой просто текстовый файл, расположенный где-либо на Web-сервере. Web-браузер, который реализует автономные приложения HTML5, прочтет список URL из файла манифеста, загрузит ресурсы, кэширует их локально и будет автоматически сохранять локальные копии по мере их изменения. Когда вы пытаетесь получить доступ к Web-приложению при отсутствии сетевого соединения, ваш Web-браузер автоматически переключается на использование локальных копий.

Начиная с этого момента, большую часть работы должны выполнить вы, как Web-разработчик. В DOM есть флаг, который сообщает вам, в каком режиме вы находитесь в данный момент — онлайн или в автономном. Существуют события, которые срабатывают при изменении вашего статуса соединения (когда вы только что были в автономном режиме, а через минуту переключились в режим онлайн, и наоборот). Но если ваше приложение создает данные или сохраняет свой статус, то локальное сохранение данных при переключении в автономный режим и их синхронизация при подключении к удаленному серверу — это ваша задача (*см. главу 7*). Иными словами, HTML5 позволяет переводить ваши Web-приложения в автономный режим. Что вы будете делать после этого — целиком и полностью зависит от вас. Краткая сводка информации о поддержке автономных Web-приложений в современных браузерах и мобильных устройствах приведена в табл. 8.1.

Таблица 8.1. *Поддержка автономных Web-приложений в современных браузерах и мобильных устройствах*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
·	✓	✓	✓	·	✓	✓

Манифест кэша

Автономные Web-приложения "крутятся" вокруг файла манифеста кэша (cache manifest file). Что такое файл манифеста? Это список всех ресурсов, которые могут понадобиться вашему Web-приложению в течение того времени, когда вы не имеете сетевого подключения и работаете автономно. Чтобы запустить процесс загрузки и кэширования этих ресурсов, вам нужен файл манифеста, использующий атрибут `manifest` в своем элементе `<html>`, как показано в листинге 8.1.

Листинг 8.1. Примерные ссылки на файл файла манифеста кэша

```
<!DOCTYPE HTML>
<html manifest="/cache.manifest">
<body>
...
</body>
</html>
```

Ваш файл манифеста кэша может находиться где угодно на вашем Web-сервере, но он должен обслуживаться с помощью MIME-типа `text/cache-manifest`. Если вы работаете с Apache или одной из производных Apache, то вы можете просто добавить директиву `AddType` (**http://httpd.apache.org/docs/2.2/mod/mod_mime.html#addtype**) в файл `.htaccess` в корневом каталоге вашего Web-сервера:

```
AddType text/cache-manifest .manifest
```

Затем вам необходимо убедиться, что имя вашего файла манифеста кэша заканчивается строкой `.manifest`. Если вы пользуетесь другим Web-сервером или другой конфигурацией Apache, сверьтесь с вашей документацией относительно управлением содержимым заголовка `Content-Type`.

Вопрос к Профессору Разметки

В: Мое Web-приложение состоит из нескольких страниц. Мне нужен атрибут `manifest` на каждой странице или достаточно иметь его только на домашней странице?

О: Каждая Web-страница вашего приложения нуждается в атрибуте `manifest`, который указывает на манифест кэша для всего приложения.

Отлично, теперь каждая из ваших HTML-страниц указывает на ваш файл манифеста кэша, и ваш файл манифеста кэша обслуживается с помощью надлежащего заголовка `Content-Type`. Но что попадает в сам файл манифеста? Именно в этот момент ситуация как раз и становится интересной.

Первая строка каждого файла манифеста кэша выглядит так:

```
CACHE MANIFEST
```

Далее, все файлы манифестов разделяются на три части: "явный" (explicit) раздел, "резервный" (fallback) раздел или раздел "онлайнового белого списка" (online whitelist). Каждый раздел имеет заголовок, идущий отдельной строкой. Если файл манифеста не имеет никаких заголовков раздела, подразумевается, что все перечисленные в нем ресурсы принадлежат к "явному" разделу.

> ***Совет***
>
> Старайтесь не "зацикливаться" на терминологии, иначе у вас просто распухнет голова.

В листинге 8.2 приведен пример корректного файла манифеста. В нем перечислены три ресурса: CSS-файл, файл на JavaScript и рисунок в формате JPEG.

Листинг 8.2. Пример корректного файла манифеста кэша

```
CACHE MANIFEST
/clock.css
/clock.js
/clock-face.jpg
```

Этот файл манифеста кэша не имеет заголовков разделов, поэтому все перечисленные в нем ресурсы по умолчанию считаются "явными". Русурсы, перечисленные в "явном" разделе, будут загружены и кэшированы локально, и при отсутствии подключения к Интернету они будут использоваться вместо своих онлайновых "дубликатов", которые находятся на сервере. Таким образом, после загрузки этого файла манифеста кэша, ваш браузер будет загружать файлы `clock.css`, `clock.js` и `clock-face.jpg` из корневого каталога вашего Web-сервера. Затем, если вы физически отключите компьютер от сети и обновите страницу, все эти ресурсы будут доступны и в автономном режиме.

> ***Вопрос к Профессору Разметки***
>
> **В**: Нужно ли мне перечислять в манифесте кэша мои HTML-страницы?
>
> **О**: И да, и нет. Если все ваше Web-приложение состоит из единственной страницы, достаточно просто убедиться в том, что она указывает на манифест кэша с использованием атрибута `manifest`. Когда вы переходите на HTML-страницу с атрибутом `manifest`, подразумевается, что сама страница является частью Web-приложения, так что вам не нужно перечислять ее в файле манифеста. Но если ваше приложение включает множество страниц, то вы должны перечислить все эти страницы в файле манифеста, в противном случае браузер не узнает о том, что есть и другие HTML-страницы, которые он тоже должен загрузить и кэшировать.

Сетевые разделы файла манифеста

Рассмотрим чуть более сложный пример. Предположим, что ваше приложение "часы" должно отслеживать посетителей, используя скрипт `tracking.cgi`, который динамически загружается из атрибута `<img src>`. Кэширование этого атрибута

"сведет на нет" цель отслеживания как таковую, поэтому данный ресурс не должен кэшироваться и никогда не должен быть доступен в автономном режиме. Как дать соответствующие указания браузеру, показано в листинге 8.3.

Листинг 8.3. Пример файла манифеста кэша с разделом NETWORK

```
CACHE MANIFEST
NETWORK:
/tracking.cgi
CACHE:
/clock.css
/clock.js
/clock-face.jpg
```

Файл манифеста кэша, представленный в листинге 8.3, содержит заголовки разделов. Заголовок `NETWORK:` представляет собой начало раздела "онлайнового белого списка". Ресурсы, перечисленные в этом разделе, никогда не кэшируются и никогда не бывают доступными в автономном режиме. Попытка их загрузки в автономном режиме приведет к ошибке. Строка `CACHE:` является началом "явного" раздела. Остальная часть этого файла манифеста кэша совпадает с ранее приведенным примером (см. листинг 8.2). Все три перечисленных в нем ресурса будут кэшированы и доступны в автономном режиме.

Резервные разделы файла манифеста

Существует еще один тип раздела файла манифеста кэша: "резервный" раздел (fallback section). В этом разделе вы определяете замены для онлайновых ресурсов, которые по любой причине не могли быть кэшированы или не были кэшированы успешно. Спецификация HTML5 предлагает пример использования "резервного раздела", приведенный в листинге 8.4.

Листинг 8.4. Пример раздела FALLBACK, предлагаемый в спецификации HTML5

```
CACHE MANIFEST
FALLBACK:
/ /offline.html
NETWORK:
*
```

Что делает этот раздел? Во-первых, вспомните о сайтах, которые содержат миллионы страниц — например, о таких, как Wikipedia (**https://secure.wikimedia.org/wikipedia/en/wiki/Main_Page**). Скорее всего, вам и не требуется загружать весь сайт целиком, да вам и не захочется этого делать. Но представьте себе, что вам хочется, чтобы некоторая часть такого сайта была доступна автономно. Но как вы примете решение о том, какие страницы следует кэшировать? Как насчет следующего подхода: каждая страница, которую вы просматриваете на некоей гипотети-

ческой Википедии с возможностью автономного доступа должна быть загружена и кэширована. Тогда вы кэшируете каждую статью энциклопедии, которую вы когда-либо посетили, каждую страницу с обсуждением (где вы можете вести дискуссии о конкретной статье энциклопедии) и каждую страницу редактирования (где вы можете фактически вносить редакторские изменения в конкретную статью).

Манифест кэша, представленный в листинге 8.4, выполняет как раз эту задачу. Представьте себе, что каждая страница HTML (статья, страница обсуждения, страница редактирования, страница истории) на сайте Wikipedia указывает на этот файл манифеста кэша. Когда вы посещаете любую страницу, которая указывает на манифест кэша, ваш браузер замечает: "Ага, эта страница — часть автономного Web-приложения, а я о ней знаю?" Если ваш браузер никогда раньше не загружал этот конкретный файл манифеста кэша, он создаст новый автономный "прикладной кэш" (сокращенно — "appcache"), загрузит все ресурсы, перечисленные в манифесте кэша, а затем добавит в кэш текущую страницу. Если ваш браузер уже "знает" об этом манифесте кэша, он просто добавит текущую страницу к существующему кэшу приложения. В любом случае, страница, которую вы только что посетили, окажется в кэше приложения. Это крайне важно. Это означает, что в вашем распоряжении окажется автономное Web-приложение, которое будет "неторопливо" добавлять страницы в кэш по мере того, как вы будете их посещать. И вам не потребуется вручную добавлять в ваш манифест кэша все HTML-страницы, которые вы посещаете.

Теперь посмотрите на "резервный" раздел. Резервный раздел в этом манифесте кэша представляет собой всего лишь единственную строку. Первая часть строки (перед пробелом) — не URL. На самом деле это шаблон URL. Единственный символ наклонной косой черты (/) будет соответствовать любой странице на вашем сайте, а не только домашней странице. Когда вы пытаетесь посетить страницу, находясь в автономном режиме, ваш браузер будет искать ее в кэше приложения. Если браузер найдет ее в кэше приложения (в том случае, если вы посещали эту страницу, когда имели интернет-соединение, и страница была неявно добавлена в кэш приложения в тот момент), то ваш браузер отобразит кэшированную копию этой страницы. Если ваш браузер не найдет копию страницы в кэше прилжения, то вместо сообщения об ошибке он отобразит страницу `/offline.html`, как указано во второй половине строки, составляющей "резервный" раздел.

Наконец, давайте изучим сетевой раздел. Сетевой раздел в этом файле манифеста кэша тоже состоит из единственной строки, которая содержит единственный символ (`*`). В сетевом разделе этот символ имеет специальное значение. Он называется "флагом шаблона онлайнового белого списка". Это — просто причудливый способ сказать, что все, что не находится в кэше приложения, может быть загружено с оригинального Web-адреса, как только появляется интернет-соединение. Это очень важно для автономного Web-приложения "с открытым концом". Это значит, что пока вы просматриваете гипотетический сайт, в чем-то похожий на Wikipedia, ваш браузер будет извлекать изображения, видео и другие встроенные ресурсы обычным путем, даже если они находятся в другом домене. Это обычная практика для масштабных Web-сайтов, даже если они не являются частью автономного Web-

приложения. HTML-страницы генерируются и обслуживаются локально, в то время, как изображения и видео поставляются из сети доставки контента (Content Delivery Network, CDN[1]) в другом домене. Без этого флага шаблона наш гипотетический сайт с возможностями Wikipedia вел бы себя странно, когда вы находитесь в оперативном режиме — то есть он не загружал бы изображения или видео с внешнего хостинга!

Можно ли считать этот пример полным? Нет. Wikipedia — это больше, чем просто HTML-файлы. Wikipedia использует общие CSS, JavaScript и изображения на каждой странице. Каждый из этих ресурсов понадобилось бы явным образом перечислять в разделе `CACHE:` файла манифеста, чтобы страницы отображались и надлежащим образом вели себя в автономном режиме. Но смысл "резервного" раздела заключается в том, что у вас будет автономное Web-приложение "с открытым концом", которое расширяется за пределы ресурсов, которые вы явным образом перечислили в файле манифеста.

Поток событий

До сих пор я говорил об автономных Web-приложениях, манифесте кэша и автономном кэше приложения ("appcache"), пользуясь неопределенными терминами, наполовину напоминающими магические заклинания. Ресурсы загружаются, браузеры сами принимают решения, и вообще — "все просто работает". В общем, вы меня уже поняли... Мы говорим о Web-разработке, не так ли? А любой разработчик знает, что ни одна вещь не будет ни с того ни с сего "просто работать".

Во-первых, давайте поговорим о ходе событий, а конкретно — событий DOM. Когда ваш браузер открывает страницу, которая указывает на манифест кэша, он запускает целую цепочку событий, происходящих с объектом `window.applicationCache`. Я понимаю, что все сейчас будет выглядеть сложно, но поверьте, что данное объяснение — это самое простое, что я смог для вас придумать, не упускающее никакой важной информации.

1. Как только браузер обнаружит атрибут `manifest` в элементе `<html>`, он запускает событие проверки `checking` (все события, упоминаемые здесь, происходят с объектом `window.applicationCache`). Событие `checking` запускается всегда, независимо от того, посещали ли вы раньше эту или другую страницу, указывающую на тот же самый манифест кэша.
2. Если ваш браузер никогда раньше "не видел" этот манифест кэша, то он:
   - Запустит событие `downloading`, затем начнет загрузку ресурсов, перечисленных в манифесте кэша.
   - Пока идет загрузка, ваш браузер будет периодически запускать события `progress`, которые содержат информацию о том, какое количество файлов уже загружено и сколько файлов осталось еще загрузить.
   - После того как будут успешно загружены все ресурсы, перечисленные в манифесте кэша, браузер запустит заключительно событие, `cached`. Это собы-

[1] См. **http://en.wikipedia.org/wiki/Content_delivery_network**.

тие служит сигналом о том, что автономное Web-приложение полностью кэшировало ресурсы и готово к автономному использованию. Иначе говоря, все готово.

3. С другой стороны, если вы раньше посещали эту или любую другую страницу, которая указывает на тот же самый манифест кэша, то вашему браузеру уже известно об этом манифесте кэша. Некоторые из ресурсов могут уже быть загружены в кэш приложения. В этом кэше могут находиться уже все ресурсы, необходимые для автономной работы приложения. Поэтому вопрос стоит так: изменился ли манифест с того момента, когда браузер проверял его последний раз?
   - Если ответ отрицателен, то есть манифест кэша не изменялся, ваш браузер немедленно запустит событие `noupdate`. Иными словами, больше вам делать ничего не нужно.
   - Если ответ положителен, это значит, что манифест кэша изменился, и ваш браузер запустит событие `downloading` и начнет повторную загрузку каждого отдельного ресурса, перечисленного в манифесте кэша.
   - Пока происходит загрузка, ваш браузер будет периодически запускать события `progress`, сообщающие о том, сколько файлов уже загружено и сколько файлов еще осталось загрузить.
   - После того как все ресурсы, перечисленные в манифесте кэша, будут успешно перезагружены, браузер запустит последнее событие, `updateready`. Это событие служит сигналом о том, что ваше автономное Web-приложение полностью обновлено и готово к автономному использованию. Новая версия при этом еще не используется. Чтобы "переключиться" на новую версию, не заставляя при этом пользователя перезагружать страницу, можно вручную вызвать функцию `window.applicationCache.swapCache()`.

Если на любой из стадий этого процесса что-то пойдет катастрофически не так, ваш браузер запустит событие `error` и остановится. Вот сокращенный до самого предельного минимума список вещей, которые могут пойти "не так":

- ❒ Манифест кэша возвращает ошибку HTTP 404 (Page Not Found) или 410 (Permanently Gone).
- ❒ Манифест кэша был обнаружен неизмененным, но HTML-страница, на которую указывает манифест, не может быть загружена надлежащим образом.
- ❒ Манифест кэша изменился во время выполнения обновления.
- ❒ Манифест кэша был обнаружен, он изменился, но браузер не смог загрузить ресурсы, перечисленные в манифесте.

Искусство отладки, или "Убейте меня! Сейчас же!"

Здесь я хочу особо подчеркнуть два важных момента. О первом из них вы уже прочли, но я готов поспорить на что угодно, что вы еще не уяснили его себе как следует, так что я повторюсь. Если хотя бы один из ресурсов, перечисленных в вашем файле манифеста кэша, не будет загружен правильно, то провалом завершится

и весь процесс кэширования вашего автономного приложения. Ваш браузер запустит событие `error`, но никакого указания на реальную причину возникновения проблемы не последует. Это делает отладку автономных Web-приложений еще более сложной задачей, чем обычную отладку.

Второй важный момент — это то, что с технической точки зрения, ошибкой не является, но будет производить впечатление серьезного бага браузера, до тех пор, пока вы не поймете, что происходит на самом деле. Связано это с тем, как ваш браузер производит проверку изменений файла манифеста кэша. Это трехэтапный процесс. Описание достаточно утомительно, но понять этот процесс очень важно, так что возьмите на себя труд и прочите его внимательно.

1. В соответствии с нормальной семантикой HTTP, ваш браузер будет проверять, пригоден ли к использованию манифест кэша. Как и для любого другого файла, обслуживаемого через HTTP, ваш Web-сервер обычно включает метаданные об этом файле в заголовки ответов HTTP. Некоторые из этих заголовков HTTP (`Expires` и `Cache-Control`) сообщают вашему браузеру о том, как ему разрешено кэшировать файл, не спрашивая сервер о том, изменился ли он. Этот тип кэширования не имеет ничего общего с автономными Web-приложениями. Это происходит практически с каждой HTML-страницей, таблицей стилей, скриптом, изображением или любым другим ресурсом Web.
2. Если срок действия манифеста кэша истек (в соответствии с его заголовками HTTP), то ваш браузер запросит сервер, нет ли там новой версии, и, если окажется, что это действительно так, браузер загрузит ее. Чтобы сделать это, ваш браузер выдаст запрос http, включающий дату последней модификации манифеста кэша, которую ваш Web-сервер включил в заголовки ответа http, когда браузер последний раз загружал файл манифеста. Если Web-сервер определит, что файл манифеста с той даты не изменялся, то он просто вернет код статуса 304 (Not Modified). Повторимся, ничто из описываемого не является специфичным для автономных Web-приложений. Это происходит практически со всеми видами Web-ресурсов.
3. Если Web-сервер определит, что файл манифеста изменился с того времени, он вернет код статуса HTTP 200 (OK), за которым последует содержимое нового файла, вместе с новыми заголовками `Cache-Control` и новой датой последней модификации, так что шаги 1 и 2 в следующий раз сработают нормально. HTTP работает четко; Web-серверы всегда ведут планирование на будущее. Если ваш Web-сервер непременно должен переслать вам файл, он сделает все, чтобы убедиться в том, что ему не нужно будет отсылать файл дважды без причин. После того как браузер загрузит новый файл манифеста кэша, он сверит его содержимое с копией, которая была загружена в последний раз. Если содержимое файла манифеста кэша осталось без изменений по сравнению с последней загруженной версией, ваш браузер не будет перезагружать ресурсы, перечисленные в манифесте.

На любом из этих шагов вы можете "споткнуться" при разработке и тестировании ваших автономных Web-приложений. Например, представим себе, что вам нужно развернуть одну версию вашего файла манифеста кэша, а затем, через 10 минут, вы поймете, что вам нужно долавить к нему еще один ресурс. Никаких

проблем, правада же? Надо просто добавить в файл еще строку и выполнить повторно развертывания. Брр. Вот что случится: вы повторно загрузите страницу, ваш браузер обнаружит атрибут `manifest`, запустит событие `checking`, и... и ничего не произойдет. Ваш браузер будет упрямо настаивать на том, что манифест кэша не изменился. Но почему? Потому, что по умолчанию ваш Web-сервер, возможно, сконфигурирован так, чтобы инструктировать браузеры, чтобы они кэшировали статические файлы на несколько часов (с помощью семантики HTTP, используя заголовки `Cache-Control`). Это значит, что ваш браузер никогда не пройдет дальше первого шага только что описанной трехшаговой процедуры. Естественно, Web-серверу известно, что файл изменился, но ваш браузер его об этом просто не спрашивает. Почему? Потому что, когда ваш браузер в последний раз загружал манифест кэша, Web-сервер сообщил ему кэшировать ресурс на несколько часов (с помощью семантики HTTP, с использованием заголовков `Cache-Control`). И теперь, через 10 минут, браузер именно так себя и ведет.

Если быть совсем уж точным, то это вовсе не баг, это "особенность" (feature), или, как принято говорить на компьютерном сленге, "фича"[1]. Все работает именно так, как и предполагалось. Если бы Web-серверы не имели возможности сообщать браузерам (и промежуточным прокси-серверам) о необходимости кэширования, то вся Паутина просто обрушилась бы. Но вам от этого не легче, особенно после того, как вы потратите несколько часов на попытки выяснить, почему браузер не замечает того, что вы обновили манифест кэша. Или еще того лучше, если вы прождете достаточно долго, все загадочным образом заработает опять! Просто потому что истечет срок годности кэша HTTP! Точно так, как и предполагалось! Убейте меня! Немедленно!

Поэтому есть одна вещь, которую вам абсолютно необходимо сделать: переконфигурировать ваш Web-сервер так, чтобы ваш файл манифеста кэша не кэшировался бы с помощью семантики HTTP. Если вы работаете с Apache или каким-нибудь еще Web-сервером на основе Apache, вам для этого потребуется добавить в файл `.htaccess` всего две строки, показанные в листинге 8.5.

Листинг 8.5. Строки, которые необходимо добавить в файл .htaccess для отключения кэширования файла манифеста кэша с помощью семантики HTTP

```
ExpiresActive On
ExpiresDefault "access"
```

На самом деле это блокирует кэширование для каждого файла в этом каталоге и его вложенных каталогах. Вероятно, это не то, что вы хотели бы сделать в вашей производственной среде, так что вам нужно будет либо воспользоваться уточнением с помощью директивы `<Files>`, чтобы изменения затронули только ваш файл манифеста кэша, или создать вложенный каталог, в котором не будет содержаться ничего, кроме этого файла `.htaccess` и вашего файла манифеста кэша. Как обычно, детали конфигурации могут варьироваться в зависимости от Web-сервера, так что

[1] См. **http://tinyurl.com/49kkoal**.

сверьтесь с документацией к вашему Web-серверу и уточните, как следует контролировать заголовки кэширования HTTP.

После того как вы блокировали кэширование HTTP для самого файла манифеста кэша, время от времени будут возникать ситуации, когда вы поменяете один или два ресурса в кэше приложения, но он по-прежнему будет оставаться под тем же URL на вашем Web-сервере. Здесь заминка происходит на шаге 2 нашего трехэтапного процесса. Если ваш файл манифеста кэша не изменился, то браузер никогда не заметит того, что изменился один из кэшированных ресурсов. Рассмотрим пример, приведенный в листинге 8.6.

Листинг 8.6. Пример, демонстрирующий ситуацию, когда браузер не замечает изменений, потому что сам файл манифеста кэша не изменился

```
CACHE MANIFEST
# rev 42
clock.js
clock.css
```

Если вы измените файл `clock.css` и выполните его повторное развертывание, вы не увидите изменений, потому что сам файл манифеста кэша не изменился. Каждый раз, когда вы вносите изменение в один из ресурсов вашего автономного Web-приложения, вам необходимо изменять сам файл манифеста кэша. Это можно сделать, просто изменив единственный символ. Самый простой способ сделать это, обнаруженный мной, сводится к изменению строки комментария с номером изменения. Измените номер версии в строке комментария, после чего Web-сервер вернет новый измененный файл манифеста кэша, ваш браузер заметит, что содержимое файла изменилось (листинг 8.7), и запустит процесс перезагрузки всех ресурсов, перечисленных в манифесте.

Листинг 8.7. Чтобы изменить файл манифеста кэша, достаточно изменить хоть один символ в строке комментария — пусть это будет номер версии

```
CACHE MANIFEST
# rev 43
clock.js
clock.css
```

Давайте создадим автономное приложение!

Вы, наверное, хорошо помните игру Halma, которую мы написали в *главе 4*, посвященной элементу `<canvas>` (**http://diveintohtml5.org/canvas.html#halma**), а впоследствии, в *главе 7*, усовершенствовали за счет сохранения статуса игры в постоянном локальном хранилище (**http://diveintohtml5.org/storage.html#halma**). А теперь давайте сделаем эту игру автономным приложением.

Чтобы сделать это, нам нужно написать файл манифеста, в котором будут перечислены все ресурсы, необходимые для этой игры. Насколько вы помните, игра состоит из основной HMTL-страницы, единственного файла на JavaScript, который содержит сам код игры, и… и все. Изображений у игры нет, потому что все вычерчивание было реализовано программно с использованием canvas API (**http://diveintohtml5.org/canvas.html**). Все необходимые стили CSS находятся в элементе `<style>` в начале HTML-страницы. Так что наш файл манифеста кэша будет выглядеть так, как показано в листинге 8.8.

Листинг 8.8. Файл манифеста кэша для преобразования игры Halma в автономное Web-приложение

```
CACHE MANIFEST
halma.html
../halma-localstorage.js
```

Необходимо сказать пару слов о путях. Я создал подкаталог `offline/` в каталоге `examples/`, и наш манифест кэша "живет" внутри вложенного каталога. Поскольку HTML-странице потребуется одно небольшое изменение, чтобы она могла работать в автономном режиме (до этого изменения мы доберемся меньше, чем через минуту), я создал отдельную копию HTML-файла, которая тоже находится в подкаталоге `offline/`. Но так как в сам код на JavaScript изменений вносить не надо, потому что мы добавили в него поддержку локального хранилища (**http://diveintohtml5.org/storage.html#halma**), я фактически повторно использую один и тот же `.js`-файл, который находится в родительском каталоге (`examples/`). Вся вместе эта файловая структура выглядит так, как показано в листинге 8.9.

Листинг 8.9. Общая файловая структура примера с переводом игры Halma в автономное Web-приложение

```
/examples/localstorage-halma.html
/examples/halma-localstorage.js
/examples/offline/halma.manifest
/examples/offline/halma.html
```

В файле манифеста кэша (`/examples/offline/halma.manifest`) мы хотим ссылаться на два файла. Первый файл — это автономная версия HTML-страницы (`/examples/offline/halma.html`). Так как эти два файла находятся в одном и том же каталоге, в файле манифеста он указывается без какого-либо префикса. Второй файл — это файл JavaScript, который находится в родительском каталоге (`/examples/halma-localstorage.js`). Он указан в файле манифеста с использованием относительной нотации URL: `../halma-localstorage.js`. Точно таким же образом вы указывали бы относительный URL в атрибуте `<img src>`. Как вы увидите в следующем примере, можно использовать и абсолютные пути (начинающиеся в коневом каталоге текущего домена) или даже абсолютные URL (указывающие на ресурсы в других доменах).

Теперь в наш HTML-файл нужно добавить атрибут `manifest`, который указывает на файл манифеста кэша. Это делается так, как показано в листинге 8.10.

Листинг 8.10. Добавление атрибута `manifest` в HTML-файл автономного Web-приложения

```
<!DOCTYPE html>
<html lang="en" manifest="halma.manifest">
```

Вот и все! Когда наш браузер, обладающий возможностями поддержки автономных Web-приложений, впервые загрузит HTML-страницу с активизированной поддержкой автономной работы (**http://diveintohtml5.org/examples/offline/halma.html**), он загрузит файл манифеста кэша, на который дана ссылка, а затем начнет загрузку все перечисленных ресурсов и сохранит их в кэше автономного Web-приложения. Начиная с этого момента, алгоритм автономного приложения будет начинать работу каждый раз, когда вы вновь посетите страницу. Вы сможете играть в игру автономно, и статус игры будет запоминаться локально, а вы сможете покидать страницу и возвращаться на нее так часто, как вам хочется.

Материалы для дальнейшего чтения

Стандарты:

- Автономные Web-приложения в спецификации HTML5 — **http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html**

Документация от поставщиков браузеров:

- Документация по автономным приложениям Центра разработки Firefox — **https://developer.mozilla.org/En/Offline_resources_in_Firefox**
- Документация по кэшу автономных Web-приложений HTML5, часть руководства разработчика Safari по клиентскому кэшу и автономным приложениям — **http://developer.apple.com/library/safari/#documentation/iPhone/Conceptual/SafariJSDatabaseGuide/OfflineApplicationCache/OfflineApplicationCache.html**

Руководства и демонстрации:

- Gmail для мобильного HTML5 (часть 1) — **http://googlecode.blogspot.com/2009/04/gmail-for-mobile-html5-series-using.html**
- Gmail для мобильного HTML5 (часть 2) — **http://googlecode.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-2.html**
- Gmail для мобильного HTML5 (часть 3) — **http://googlecode.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-3.html**
- Статья Джонатана Старка (Jonathan Stark) *"Debugging HTML5 offline application cache"* — **http://jonathanstark.com/blog/2009/09/27/debugging-html-5-offline-application-cache/**
- Демонстрационный пример автономного Web-редактора изображений от разработчиков Mozilla — **https://hacks.mozilla.org/2010/02/an-html5-offline-image-editor-and-uploader-application/**

Глава 9

Форма безумия

Каждый знает о Web-формах, не так ли? Создайте элемент `<form>`, несколько элементов `<input type="text">`, может быть даже элементов `<input type="password">`, завершите все это кнопкой `<input type="submit">`, вот и готово!

Если вы знаете только это, значит, вы не знаете и половины того, что вам необходимо знать. HTML5 определяет более дюжины новых типов ввода, которые вы можете использовать в своих формах. И когда я говорю "использовать", я имею в виду, что вы можете начинать ими пользоваться прямо сейчас — без каких бы то ни было "хаков", ухищрений и обходных путей. Но не надо чересчур волноваться — я не сказал, что все эти новые возможности немедленно будут поддерживаться каждым браузером. О нет, я этого совсем не говорил. В современных браузерах да, ваши формы могут "задать жару" кому угодно и делать практически что угодно. Но вот в более старых браузерах, хотя ваши формы и будут работать, они продемонстрируют меньше впечатляющих возможностей. Впрочем, надо заметить, что это снижение функциональных возможностей во всех браузерах происходит довольно деликатно, даже в IE 6.

Замещающий текст

Первое усовершенствование, которое HTML5 вносит в Web-формы — это возможность использования замещающего текста (placeholder text) в поле ввода (**http://www.whatwg.org/specs/web-apps/current-work/multipage/common-input-element-attributes.html#the-placeholder-attribute**). Замещающий текст отображается внутри поля ввода до тех пор, пока поле является пустым и не имеет фокуса ввода (рис. 9.1). Как только вы щелкнете на этом поле или перейдете в него нажатием клавиши <Tab>, замещающий текст исчезает (рис. 9.2).

Рис. 9.1. Замещающий текст отображается внутри поля ввода до тех пор, пока поле является пустым и не имеет фокуса ввода

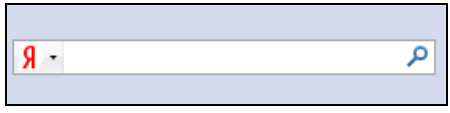


Рис. 9.2. Как только поле ввода получает фокус ввода, замещающий текст исчезает

Краткая информация о поддержке замещающего текста в полях ввода современными браузерами и мобильными устройствами приведена в табл. 9.1.

Таблица 9.1. *Поддержка замещающего текста в полях ввода в современных браузерах и мобильных устройствах*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	3.7+	4.0+	4.0+	.	4.0+	.

Вероятно, вы уже много раз видели замещающий текст, например, в Mozilla Firefox (см. рис. 9.1).

В листинге 9.1 показано, как можно включить замещающий текст в ваши собственные Web-формы.

Листинг 9.1. Включение замещающего текста в Web-форму

```
<form>
  <input name="q" placeholder="Search Bookmarks and History">
  <input type="submit" value="Search">
</form>
```

Браузеры, которые не поддерживают замещающего текста, будут игнорировать соответствующий атрибут. Нет ущерба — нет и нарушения. Чтобы проверить, поддерживает ли ваш браузер замещающий текст, пройдите по следующей ссылке: **http://diveintohtml5.org/examples/input-placeholder.html**.

Вопрос к Профессору Разметки

В: Могу ли я использовать разметку HTML в атрибуте `placeholder`? Я хочу вставить туда изображение или, может быть, даже изменить цвета.

О: Атрибут `placeholder` может содержать только текст, но не разметку HTML. Однако существуют некоторые расширения CSS, предоставляемые конкретными поставщиками (**http://trac.webkit.org/export/37527/trunk/LayoutTests/fast/forms/placeholder-pseudo-style.html**), которые позволяют вам накладывать стили на замещающий текст в некоторых браузерах.

Поля автофокуса

Краткая сводка информации о поддержке функции полей автофокуса современными браузерами и мобильными устройствами приведена в табл. 9.2.

Таблица 9.2. *Поддержка полей автофокуса в современных браузерах и мобильных устройствах*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	.	4.0+	3.0+	10.0+	.	.

Web-сайты могут использовать JavaScript для автоматической установки фокуса ввода на первом поле Web-формы. Например, домашняя страница сайта Google (**http://www.google.com/**) автоматически установит фокус на поле ввода так, чтобы, зайдя на сайт, вы могли сразу же начать ввод ключевых слов для поиска. Хотя для большинства людей это удобно, некоторых, в частности, опытных пользователей или людей, имеющих специфические потребности, это может раздражать. Если вы нажмете, например, клавишу пробела и ожидаете, что в результате будет выполнена прокрутка страницы, этого не произойдет, потому что фокус ввода уже автоматически установлен на поле ввода формы. Вместо прокрутки в поле ввода будет введен символ пробела. Если вы передаете фокус ввода другому полю, пока происходит загрузка страницы, скрипт автоматической фокусировки сайта "услужливо" переместит фокус ввода обратно исходному полю ввода, что отвлечет вас от ваших задач, потому что вы в результате начнете вводить текст не в то поле.

Поскольку автоматическая фокусировка осуществляется на JavaScript, все эти крайние случаи могут оказаться сложны в обработке, и у людей, которые не хотят, чтобы Web-страница "похищала" фокус ввода, будет очень мало возможностей отказаться от этого "сервиса".

Чтобы решить эту проблему, HTML5 вводит атрибут `autofocus` для всех элементов управления Web-форм (**http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#autofocusing-a-form-control**). Атрибут `autofocus` делает именно то, что можно предположить по его названию: как только страница загружается, он передает фокус ввода в конкретное поле ввода. Но, поскольку это не скрипт, а всего лишь разметка, это поведение будет непротиворечивым на всех Web-сайтах. Кроме того, поставщики браузеров (или разработчики расширений) могут предложить пользователям возможности по блокировке функции автоматической фокусировки.

Установку автоматической фокусировки на конкретном поле Web-формы можно выполнить так, как показано в листинге 9.2.

Листинг 9.2. Установка автоматической фокусировки на конкретном поле Web-формы

```
<form>
  <input name="q" autofocus>
  <input type="submit" value="Search">
</form>
```

Браузеры, которые не поддерживают атрибута автоматической фокусировки, будут его игнорировать. Хотите проверить, поддерживает ли ваш браузер автоматическую фокусировку? Для этого пройдите по следующей ссылке: **http://diveintohtml5.org/examples/input-autofocus.html**.

Что? Вы хотите, чтобы поля с автоматической фокусировкой работали во всех браузерах, а не только в этих "навороченных" современных монстрах, поддерживающих HTML5? Вы можете сохранить тот скрипт автоматической фокусировки, которым вы пользуетесь на данн момент. Просто внесите два небольших изменения:

- Добавьте атрибут `autofocus` к вашей разметке HTML.
- Проверьте, поддерживает ли браузер атрибут autofocus (о том, как это делается, рассказывалось в *главе 2*), и запускайте скрипт автоматической фокусировки только тогда, когда браузер не обеспечивает встроенной поддержки функции автоматической фокусировки.

Реализация автоматической фокусировки с резервными возможностями показана в листинге 9.3.

Листинг 9.3. Функция автоматической фокусировки с резервными возможностями

```
<form name="f">
  <input id="q" autofocus>
  <script>
    if (!("autofocus" in document.createElement("input"))) {
      document.getElementById("q").focus();
    }
  </script>
  <input type="submit" value="Go">
</form>
…
```

Пример реализации функции автоматической фокусировки с резервными возможностями показан здесь: **http://diveintohtml5.org/examples/input-autofocus-with-fallback.html**.

Как передать фокус по возможности раньше

Многие Web-страницы не устанавливают фокус до тех пор, пока не сработает событие `window.onload`. Но событие `window.onload` не сработает до тех пор, пока не загрузятся все изображения. Если ваша страница содержит большое количество изображений, такой наивный скрипт может передать фокус ввода другому полю уже после того, как пользователь начал взаимодействовать с другой частью страницы, не дожидаясь загрузки изображений. Вот почему многие опытные пользователи просто ненавидят скрипты автоматической фокусировки.

Пример, приведенный в предыдущем разделе (см. листинг 9.3), содержал скрипт автоматической фокусировки непосредственно после поля формы, на которую

он ссылается. Это оптимальное решение, но может быть, размещение блока кода на JavaScript прямо в середине страницы оскорбляет ваши чувства. Или, если говорить более прозаическим языком, может быть, ваши серверные системы просто недостаточно чувствительны. Если вы не можете вставить скрипт в середину страницы, вам следует установить фокус, когда происходит индивидуальное событие наподобие события jQuery `$(document).ready()`[1] вместо события `window.onload`.

Пример, иллюстрирующий автоматическую фокусировку с помощью jQuery, приведен в листинге 9.4.

Листинг 9.4. Автоматическая фокусировка с помощью jQuery

```
<head>
<script src=jquery.min.js></script>
<script>
  $(document).ready(function() {
    if (!("autofocus" in document.createElement("input"))) {
      $("#q").focus();
    }
  });
</script>
</head>
<body>
<form name="f">
  <input id="q" autofocus>
  <input type="submit" value="Go">
</form>
```

Просмотреть действующий пример использования autofocus с резервом в качестве jQuery можно по следующей ссылке: **http://diveintohtml5.org/examples/input-autofocus-with-fallback-document-ready.html**.

jQuery запускает свое индивидуальное событие, сигнализирующее о готовности, как только доступна DOM страницы — иначе говоря, jQuery ждет до тех пор, пока не загрузится текст страницы, но не дожидается загрузки всех изображений. Это не оптимальный подход — если страница слишком велика или сетевое соединение слишком медленно, то пользователь может начать взаимодействие со страницей до того, как скрипт начнет исполнение. Но это все равно лучше, чем ждать срабатывания события `window.onload`.

Если вы хотите и можете вставить единственное скриптовое утверждение в разметку вашей страницы, то есть промежуточный подход, который менее неудобен, чем первый вариант, но лучше, чем второй. Вы можете использовать индивидуальные события jQuery, чтобы определить ваше собственное событие, например, `autofocus_ready`. Затем вы можете запускать это событие вручную, сразу же после

[1] См. **http://tinyurl.com/l9umjs**.

того, как станет доступным поле автоматической фокусировки. Я благодарен Е. М. Штернбергу (E. M. Sternberg), научившему меня этой технике.

Пример использования автоматической фокусировки с использованием индивидуальных событий приведен в листинге 9.5.

Листинг 9.5. Автоматическая фокусировка с использованием индивидуальных событий в качестве резерва

```
<head>
<script src=jquery.min.js></script>
<script>
  $(document).bind('autofocus_ready', function() {
    if (!("autofocus" in document.createElement("input"))) {
      $("#q").focus();
    }
  });
</script>
</head>
<body>
<form name="f">
  <input id="q" autofocus>
  <script>$(document).trigger('autofocus_ready');</script>
  <input type="submit" value="Go">
</form>
```

Пример использования `autofocus` с индивидуальным событием показан здесь: **http://diveintohtml5.org/examples/input-autofocus-with-fallback-custom-event.html**.

В качестве первого приближения этот подход оптимален; он установит фокус на поле формы сразу же, как это станет технически возможно, пока текст страницы может еще продолжать загружаться. Но при этом подходе большая часть логики вашего приложения (передача фокуса ввода полю формы) переходит из тела страницы в ее заголовок. Этот пример полагается на jQuery, но концепция индивидуальных событий не уникальна для jQuery. Другие библиотеки JavaScript наподобие YUI и Dojo тоже предлагают аналогичные возможности.

Подводя итоги, скажем:

- Правильная установка фокуса ввода очень важна.
- Если это вообще возможно, позвольте браузеру сделать это для вас, установив атрибут `autofocus` на поле формы, которому должен быть передан фокус ввода.
- Если вы кодируете резервную возможность для более старых браузеров, выявляйте поддержку атрибута `autofocus`, чтобы гарантировать, что ваш резервный код будет исполняться только в более старых браузерах.
- Устанавливайте фокус как можно раньше. Вставьте скрипт установки фокуса в разметку непосредственно за полем, которому должен быть передан фокус.

Если вы не можете себе этого позволить, используйте библиотеку JavaScript, поддерживающую индивидуальные события, и запустите индивидуальное событие сразу же после разметки поля формы. Если это не представляется возможным, используйте что-нибудь наподобие события jQuery `$(document).ready()`.

- Ни при каких обстоятельствах не следует дожидаться, пока фокус будет установлен событием `window.onload`.

Адреса электронной почты

В течение более десятка лет Web-формы содержали лишь несколько видов полей. Наиболее распространенные типы полей перечислены в табл. 9.3.

Таблица 9.3. *Наиболее распространенные типы полей в Web-формах*

Тип поля	Код HTML	Примечания
Флажок (checkbox)	`<input type="checkbox">`	Может быть либо установлен, либо сброшен
Переключатель (radio button)	`<input type="radio">`	Может группироваться с другими полями
Поле ввода пароля (password field)	`<input type="password">`	По мере того как вы вводите в это поле пароль, поле отображает вместо вводимых символов точки
Раскрывающиеся списки (drop-down lists)	`<select><option>…`	
Выбор файла (file picker)	`<input type="file">`	Приводит к появлению окна **Open File**
Кнопка **Submit**	`<input type="submit">`	
Обычный текст (plain text)	`<input type="text">`	Атрибут типа может быть пропущен

Все эти типы полей ввода по-прежнему будут работать в HTML5. Если вы выполняете "обновление до HTML5", возможно, просто изменяя `DOCTYPE` (**http://diveintohtml5.org/semantics.html#the-doctype**), то вам нет необходимости вносить единственное исправление в ваши Web-формы. Ура! Да здравствует обратная совместимость!

Но HTML5 определяет 13 новых типов полей, и по ряду причин, которые станут вам ясны буквально через несколько минут, нет никакой причины ими *не* пользоваться.

Первым типом новых полей ввода являются поля адреса электронной почты (**http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#e-mail-state**). Поля эти выглядят так, как показано в листинге 9.6.

Листинг 9.6. Поле адреса электронной почты

```
<form>
  <input type="email">
  <input type="submit" value="Go">
</form>
```

Я как раз собирался написать предложение "в браузерах, которые не поддерживают `type="email"`...", но тут же одернул сам себя. Почему? Потому что я не уверен в том, что точно знаю, что означает фраза о том, что "браузер не поддерживает `type="email"`...". Все браузеры "поддерживают" `type="email"`. Они могут не делать с этим полем ничего особенного (примеры особой обработки вы увидите буквально через несколько минут), но браузеры, которые не распознают `type="email"`, будут обращаться с этим полем ввода так, как если бы оно имело `type="text"` и будут отображать его как обычное текстовое поле.

Я не могу не подчеркнуть, насколько это важно. В Web есть миллионы форм, в которых вам предлагается ввести ваш почтовый адрес, и все они используют `<input type="text">`. Вы видите текстовое поле, вы вводите туда свой адрес электронной почты, и это все. И тут появляется HTML5, с типом `type="email"`. И что, это выбивает браузеры из колеи? Ничего подобного. Каждый отдельный браузер обращается с неизвестным типом как с типом `type="text"` — даже IE 6. Поэтому вы можете сделать "апгрейд" своим Web-формам хоть сейчас, и хоть сию минуту начинать использовать в них `type="email"`.

Что на самом деле означает *настоящая* поддержка браузером `type="email"`? Довольно большое количество самых разных вещей. Спецификация HTML5 не навязывает для новых типов ввода каких-нибудь особых пользовательских интерфейсов. Например, Opera украшает поле формы небольшим значком, символизирующим электронную почту. Другие браузеры с поддержкой HTML5, например, Safari и Chrome, отображают его просто как обычное текстовое поле (в точности так, как поле с типом `type="text"`), так что ваши пользователи никогда не почувствуют разницы (если только не будут просматривать исходный код).

И, наконец, совершенно особняком стоят устройства iPhone.

У iPhone нет физической клавиатуры. Весь "печатный ввод" осуществляется "нажатиями" на клавиши экранной клавиатуры, которая появляется в определенные моменты, например, когда вы передаете фокус ввода полю Web-формы. Компания Apple добавила ряд интеллектуальных нововведений в Web-браузер устройств iPhone. Так, этот браузер распознает некоторые из новых типов ввода HTML5 и динамически изменяет вид экранной клавиатуры, чтобы оптимизировать возможности ввода для конкретного типа вводного поля (рис. 9.3).

Вот, например, адреса электронной почты представляют собой текстовые строки, не так ли? Естественно, это так, но они представляют собой особый вид текста. Так, все адреса электронной почты содержат символ `@` и, как минимум, один символ точки (`.`), но крайне сомнительно, чтобы они содержали некоторые другие символы (например, пробелы). Так что если вы пользуетесь iPhone и передаете

фокус ввода полю `<input type="email">`, то экранная клавиатура будет содержать символ пробела, размер которого меньше обычного, но специальные, более удобные, чем обычно клавиши с символами @ и ., как показано на рис. 9.3.

Практический пример поля `<input type="email">` можно найти по следующему адресу: **http://diveintohtml5.org/examples/input-type-email.html**.

Подводя итоги, скажем, что немедленный переход на использование полей адресов электронной почте с типом `type="email"` ни к каким проблемам не приведет. На практике, этого почти никто и не заметит, за исключением пользователей iPhone, да и они могут не обратить на это внимания, если недостаточно наблюдательны. Ну, а те, кто заметит, возможно, улыбнутся и, может быть, поблагодарят вас за то, что вы сделали их работу удобнее и приятнее.

Рис. 9.3. Экранная клавиатура iPhone, оптимизированная для ввода адреса электронной почты

Web-адреса

Web-адреса, которые строгие приверженцы следования стандартам чаще называют URL, представляют собой еще один тип специализированного текста. Синтаксис Web-адреса ограничен соответствующими стандартами Интернета. Если кто-нибудь предложит вам ввести в форму Web-адрес, то они, скорее всего, ожидают, что вы введете что-нибудь наподобие **http://www.google.com/**, а не строку вроде "125 Farwood Road". Символы наклонной косой черты (/) тоже достаточно распространены — даже на домашней странице Google вы найдете целых три. Распространены и символы точки (.), зато не разрешается использование пробелов. И еще — любой Web-адрес имеет доменный суффикс наподобие `.com` или `.org`.

Посмотрите... (фанфары и барабанный бой, пожалуйста)... `<input type="url">`. На iPhone все это выглядит примерно так, как показано на рис. 9.4.

Посмотреть пример того, как работает `<input type="url">`, можно здесь: **http://diveintohtml5.org/examples/input-type-url.html**.

На iPhone была изменена экранная клавиатура, примерно так же, как это было сделано для адресов

Рис. 9.4. Экранная клавиатура iPhone, оптимизированная для ввода Web-адреса

электронной почты, но теперь она была оптимизирована для Web-адресов. Клавиша "Пробел" теперь полностью заменена тремя виртуальными клавишами: "точкой", прямым слэшем (/) и кнопкой .com. (если вы нажмете на кнопку .com и продержите ее некоторое время нажатой, вы получите возможность выбрать некоторые другие, чуть менее распространенные суффиксы, например, .org или .net).

Браузеры, которые не поддерживают HTML5, будут обрабатывать type="url" точно так же, как type="text", так что ничто не создаст вам никаких неудобств при использовании этого типа поля ввода для ваших целей ввода информации.

Числа как счетчики с элементами прокрутки

Далее мы перейдем к рассмотрению чисел. Запрос на ввод числа сложнее, чем запрос на ввод адреса электронной почты или Web-адреса. В первую очередь, числа на самом деле намного сложнее, чем кажутся на первый взгляд. Давайте выберем какое-нибудь число быстро. –1? Нет, я хотел число от 1 до 10. 7½? Нет, мне не нужны дроби. π? Что? Да вы совершенно иррациональны!

Моя точка зрения такова, что вы не так уж часто просите пользователя ввести "какое-нибудь число". Вероятнее всего, вы запрашиваете число из конкретного диапазона. Скорее всего, вы готовы принять только вполне конкретные числа из этого диапазона — например, только целые числа, а не дроби или десятичные дроби, и не что-то отвечающее более конкретному требованию (например, числа, кратные 10). HTML5 отвечает вашим запросам.

Пример определения поля для выбора числа, (практически) любого числа показан в листинге 9.7.

Листинг 9.7. Пример определения поля для выбора числа

```
<input type="number"
       min="0"
       max="10"
       step="2"
       value="6">
```

Давайте разберем атрибуты этого поля по одному (практический пример определения числового поля можно просмотреть по следующему адресу: **http://diveintohtml5.org/examples/input-type-number-min-max-step.html**).

- type="number" — указывает, что данное поле является числовым;
- min="0" — указывает минимальное приемлемое значение для данного поля;
- max="10" — указывает максимальное приемлемое значение для данного поля;
- step="2", в комбинации со значением min, определяет допустимые числовые значения из выбранного диапазона: 0, 2, 4 и так далее, вплоть до значения max;
- value="6" — указывает значение по умолчанию. Оно должно выглядеть как что-то хорошо знакомое. Это — то же самое имя атрибута, которые вы всегда использовали для полей формы. (Я упоминаю об этом здесь, чтобы лишний раз напомнить вам о том, что HTML5 основывается на предшествующих версиях

HTML. Вам нет необходимости заново переучивать все, что вы знали до сих пор, чтобы продолжать делать то, что вы делаете и сейчас.)

Это — все, что касается разметки числового поля. Имейте в виду, что все перечисленные атрибуты являются необязательными. Если у вас есть минимум, но нет максимума, то вы можете указать атрибут `min` и не указывать атрибута `max`. Для атрибута `step` по умолчанию принимается значение 1, и вы можете пропустить этот атрибут, если не хотите менять стандартное значение шага. Если значение по умолчанию не указано, то атрибут `value` может представлять собой либо пустую строку, либо быть просто пропущен.

Но HTML5 на этом не останавливается. Помимо прочего, вы получаете в свое распоряжение следующие удобные методы JavaScript:

- `input.stepUp(n)` — увеличивает значение поля на `n`;
- `input.stepDown(n)` — уменьшает значение поля на `n`;
- `input.valueAsNumber` — возвращает текущее значение как значение с плавающей точкой (свойство `input.value` всегда является строкой).

У вас возникают проблемы с визуализацией всего этого? Что ж, точный интерфейс элемента управления с типом числового поля зависит от вашего браузера, и различные поставщики браузеров реализовали его поддержку различными способами. На устройствах iPhone браузер снова отображает экранную клавиатуру, оптимизированную для ввода чисел (рис. 9.5).

В настольных версиях Opera, то же самое поле `type="number"` визуализируется как элемент управления в виде поля с двумя небольшими "стрелками" на правой границе, щелчки по которым увеличивают или уменьшают значение, введенное в поле (рис. 9.6).

Рис. 9.5. На устройствах iPhone браузер отображает экранную клавиатуру, оптимизированную для ввода чисел

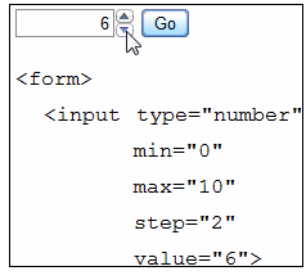


Рис. 9.6. Вид числового поля в настольной версии Opera

Рис. 9.7. При достижении максимально допустимого значения возможность дальнейшего увеличения введенного в поле числа будет заблокирована

Opera интерпретирует атрибуты `min`, `max` и `step`, так что вы в любом случае получите приемлемое числовое значение. Когда вы "натолкнетесь" на максимально допустимое значение, небольшая кнопка с изображением направленной вверх треугольной стрелки станет отображаться блекло-серым цветом (станет недоступной), как показано на рис. 9.7.

Как и в случае с остальными типами ввода, которые обсуждались ранее в этой главе, браузеры, которые не поддерживают `type="number"` будут обращаться с числовыми полями как с полями типа `type="text"`. Значение по умолчанию появится в поле (потому что оно имеет атрибут `value`), но другие атрибуты, такие, как `min` и `max`, будут игнорироваться. Вы можете реализовать их самостоятельно или можете повторно использовать инфраструктуру на JavaScript framework, в которой уже реализованы такие "счетчики". Но предварительно не забудьте проверить наличие встроенной поддержки HTML5 (см. *главу 2*), как показано в листинге 9.8.

Листинг 9.8. Проверка встроенной поддержки типа численного ввода с помощью библиотеки `Modernizr`

```
if (!Modernizr.inputtypes.number) {
  // no native support for type=number fields
  // maybe try Dojo or some other JavaScript framework
}
```

Ввод чисел с помощью ползунковых регуляторов

"Счетчики" в виде полей с двумя треугольными стрелками на правой границе не являются единственным методом числового ввода. Скорее всего, раньше вы уже видели "ползунковые регуляторы", подобные показанному на рис. 9.8.

Теперь вы можете реализовать такие элементы управления в своих формах. Рабочий пример можно посмотреть здесь: **http://diveintohtml5.org/examples/input-type-range.html**.

Рис. 9.8. Ползунковый регулятор

Разметка таких элементов управления выглядит похожей на разметку "счетчиков". Она показана в листинге 9.9.

Листинг 9.9. Разметка элемента управления `<input type="range">`

```
<input type="range"
       min="0"
       max="10"
       step="2"
       value="6">
```

Все доступные атрибуты — точно такие же, как для `type="number"` — `min`, `max`, `step`, `value`, и имеют точно такой же смысл. Единственное отличие заключается в пользовательском интерфейсе. Вместо поля для ввода браузеры отображают элемент управления type="range" в виде ползункового регулятора. На текущий момент эта функция поддерживается последними версиями Safari, Chrome и Opera. К сожалению, iPhone визуализирует этот элемент управления как простое текстовое поле. Его браузер даже не оптимизирует для этого элемента экранную клавиатуру. Все остальные браузеры обрабатывают данный элемент как `type="text"`, так что большого смысла в том, чтобы немедленно начинать его использование, пока нет.

Элементы выбора даты

В HTML 4 элементы выбора даты реализованы не были. Инфраструктуры JavaScript эту задачу решили — к их числу относятся Dojo (**http://docs.dojocampus.org/dijit/form/DateTextBox**), jQuery UI (**http://docs.jquery.com/UI/Datepicker**), YUI (**http://developer.yahoo.com/yui/calendar/**), Closure Library (**https://closure-library.googlecode.com/svn/docs/class_goog_ui_DatePicker.html**), но естественно, чтобы воспользоваться каждым из этих решений, требуется работать именно с той инфраструктурой, в которой построен элемент выбора даты.

HTML5 определяет способ включения элемента выбора даты на основе встроенных возможностей без необходимости самостоятельного написания скриптов. Фактически этих элементов определено целых шесть: `date`, `month`, `week`, `time`, `date + time` и `date + time - timezone`.

Но на сегодняшний день их поддержку браузерами можно назвать… скудной. Краткая сводка информации о поддержке элементов управления для выбора датами в современных браузерах приведена в табл. 9.4.

Таблица 9.4. *Поддержка современных элементов управления выбором даты в новейших браузерах*

Тип ввода	Opera	Остальные браузеры
`type="date"`	9.0+	.
`type="month"`	9.0+	.
`type="week"`	9.0+	.
`type="time"`	9.0+	.
`type="datetime"`	9.0+	.
`type="datetime-local"`	9.0+	.

Визуализация элемента `<input type="date">` новейшей версией Opera[1] показана на рис. 9.9. Чтобы посмотреть этот пример на практике, пройдите по ссылке: **http://diveintohtml5.org/examples/input-type-date.html**.

Если вы хотите работать с этой датой, то Opera поддерживает и элемент управления `<input type="date">` (рис. 9.10).

Практический пример можно просмотреть по следующему адресу: **http://diveintohtml5.org/examples/input-type-datetime.html**.

Если вам требуются только месяц и год (предположим, чтобы указать срок истечения годности кредитной карты), то Opera может отобразить элемент выбора даты `<input type="month">`, показанный на рис. 9.11.

Opera предоставляет и несколько менее распространенную, но тоже полезную возможность выбора конкретной недели в течение года с помощью элемента управления `<input type="week">` (рис. 9.12). Практический пример можно просмотреть, пройдя по следующей ссылке: **http://diveintohtml5.org/examples/input-type-week.html**.

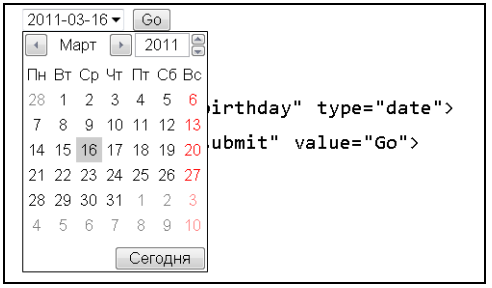


Рис. 9.9. Визуализация элемента `<input type="date">` браузером Opera 11

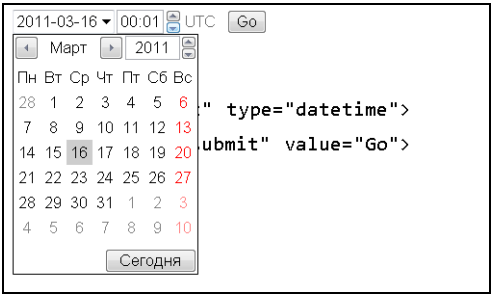


Рис. 9.10. Визуализация элемента `<input type="date">` браузером Opera 11

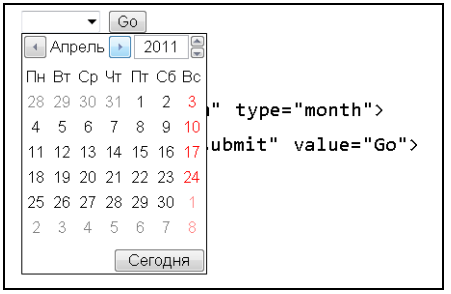


Рис. 9.11. Визуализация элемента управления для выбора даты `<input type="month">` с помощью Opera 11

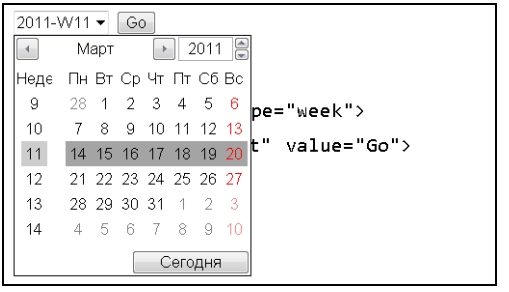


Рис. 9.12. Визуализация элемента управления выбора недели года с помощью элемента управления `<input type="week">` с помощью Opera 11

[1] На момент подготовки русского издания к печати это — версия 11 (см. **http://ru.opera.com/**). — *Прим. перев.*

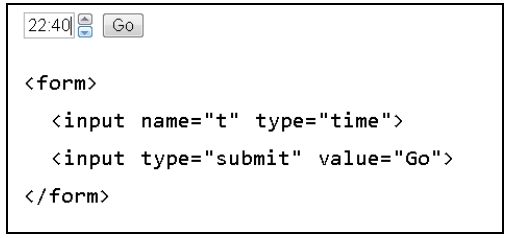


Рис. 9.13. Визуализация элемента управления для выбора времени `<input type="time">` с помощью Opera 11

Наконец, последняя, но от того не менее важная, возможность, предоставляемая Opera — это возможность выбора времени с помощью элемента управления `<input type="time">` (рис. 9.13). Практический пример можно просмотреть, пройдя по ссылке: **http://diveintohtml5.org/examples/input-type-time.html**.

Вполне возможно, что другие браузеры со временем тоже начнут поддерживать эти типы ввода. Но, как и в случае с `type="email"`, браузеры, которые пока не поддерживают новых возможностей, будут отображать эти поля как обычные поля текстового ввода. Если вы хотите, вы можете просто использовать `<input type="date">`, и Opera будет отображать их по-новому. Остальные браузеры со временем "подтянутся". Более реалистичный подход заключается в использовании `<input type="date">`, выявлении поддержки браузером элементов выбора даты (см. *главу 2*) и использовании одного из решений, применяющих скрипты на основе одной из инфраструктур JavaScript: Dojo (**http://docs.dojocampus.org/dijit/form/DateTextBox**), jQuery UI (**http://docs.jquery.com/UI/Datepicker**), YUI (**http://developer.yahoo.com/yui/calendar/**), Closure Library (**https://closure-library.googlecode.com/svn/docs/class_goog_ui_DatePicker.html**), или какую-либо другую инфраструктуру. Пример решения, использующего такой подход, приведен в листинге 9.10.

Листинг 9.10. Пример решения, использующего `<input type="date">` с резервной поддержкой на основе скриптов

```
<form>
  <input type="date">
</form>
...
<script>
  var i = document.createElement("input");
  i.setAttribute("type", "date");
  if (i.type == "text") {
    // Встроенной поддержки элемента выбора даты нет :(
    // Для создания такого интерфейсного элемента
    // можно воспользоваться Dojo/jQueryUI/YUI/Closure,
    // затем динамически заменять элемент <input>.
  }
</script>
```

Поля поиска

Это — "тонкий" момент. Хорошо, сама идея достаточно проста, но ее реализации могут потребовать некоторых объяснений. Давайте начнем разбираться.

Поиск. Не просто Google Search или Yahoo Search (хотя и они, конечно, тоже). Представьте себе просто любое поле поиска, на любой странице любого сайта. Например, такие поля имеются на Amazon, Newegg и множестве других сайтов. Большинство блогов имеют поля поиска. Как они размечаются? Да как любое другое текстовое поле в Web: `<input type="text">`. Давайте это поправим. Пример современной разметки поля поиска приведен в листинге 9.11.

Листинг 9.11. Пример разметки современного поля поиска

```
<form>
  <input name="q" type="search">
  <input type="submit" value="Find">
</form>
```

Протестируйте тип ввода `<input type="search">` (**http://diveintohtml5.org/examples/input-type-search.html**) в вашем собственном браузере. В некоторых браузерах вы не заметите никаких отличий от обычного текстового поля. Но если вы работаете с Safari в Mac OS X, то поле будет выглядеть так, как показано на рис. 9.14.

Вы можете заметить разницу? Поле ввода имеет скругленные углы! Я знаю, что некоторые с трудом могут сдержать волнение. Но это еще не все! Когда вы действительно начнете ввод в поле `type="search"`, Safari добавляет на правую границу поля небольшую круглую кнопку со значком **x**. Нажатие на этот значок очищает содержимое поля. Браузер Google Chrome, во многом реализующий технологии, похожие на используемые Safari, демонстрирует аналогичное поведение.

Оба эти небольших "трюка" сделаны для того, чтобы имитировать вид и поведение полей поиска в iTunes и других клиентских приложениях Mac OS X.

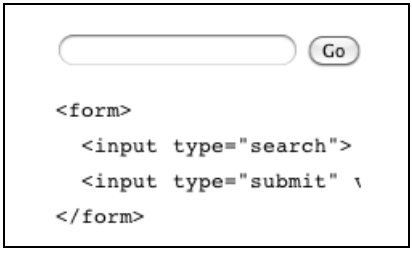


Рис. 9.14. Поле поиска в браузере Safari при работе в Mac OS X

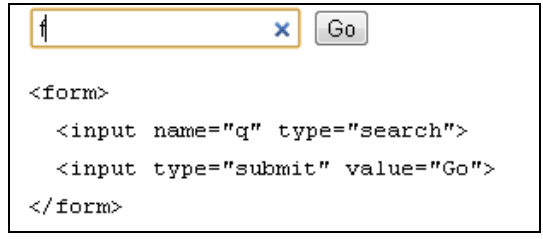


Рис. 9.15. После ввода на правой границе поля поиска в Google Chrome появляется небольшой значок x, щелчок мышью по которому очищает содержимое поля

Сайт Apple (**http://www.apple.com/**) использует `<input type="search">` для реализации полей поиска, чтобы они выглядели и вели себя "как на Mac". Но в этом нет ничего специфичного именно для Mac. Это просто разметка, поэтому любой браузер на любой платформе может визуализировать ее в соответствии с соглашениями, специфичными для этой платформы. Как и в случае с другими новыми типами ввода, браузеры, которые не интерпретируют `type="search"`, будут обращаться с ним, как с полем `type="text"`, так что большого смысла сразу же переходить на использование `type="search"` для всех ваших полей поиска пока нет.

Замечание Профессора Разметки

По умолчанию, Safari не будет применять базовых стилей CSS к полям `<input type="search">`. Если вы хотите "заставить" Safari обращаться с вашими полями поиска, как с обычными текстовыми полями (чтобы вы могли применить к ним собственные стили CSS), добавьте в вашу таблицу стилей следующее правило:

```
input[type="search"] {
  -webkit-appearance: textfield;
}
```

Я благодарен Джону Лейну (John Lein), подсказавшему мне этот "трюк".

Элементы выбора цвета

HTML5 включает и типы ввода `<input type="color">` (**http://www.whatwg.org/specs/web-apps/current-work/multipage/number-state.html#color-state**), который позволяет вам выбирать цвет и возвращает его шестнадцатеричное представление.

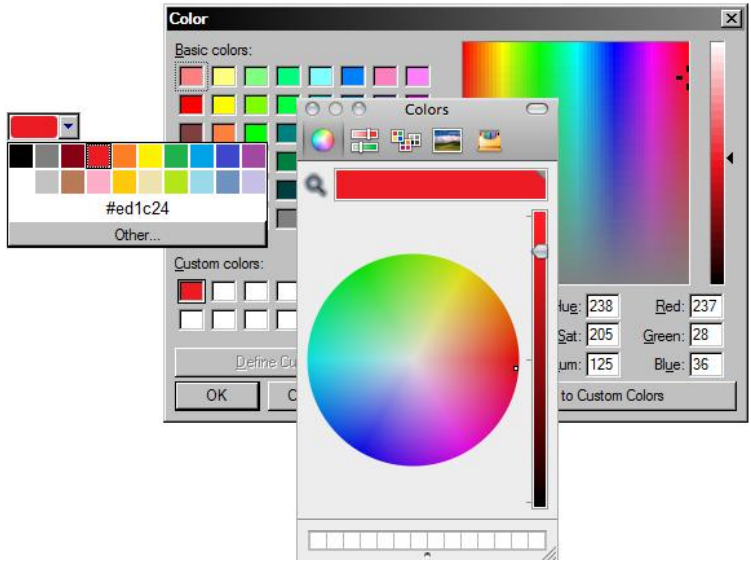


Рис. 9.16. Opera 11 теперь поддерживает `<input type="color">`

До последнего времени ни один браузер этой возможности не поддерживал, к глубокому моему сожалению, потому что я всегда любил элемент выбора цвета в стиле Mac OS X (**http://earthlingsoft.net/ssp/blog/2006/04/colour_me_picked**). Но за последнее время появились и хорошие новости! Теперь этот тип ввода поддерживается в Opera 11 (**http://dev.opera.com/articles/view/new-form-features-in-html5/#input-color**).

На платформах Mac и Windows, Opera интегрирует его со встроенным инструментом выбора цвета, специфичным для платформы. В Linux браузер отбрасывает базовый инструмент выбора цвета. Причем на всех платформах отображается значение входного элемента в шестнадцатеричном представлении цветов RGB, пригодном для обрамления и использования везде, где принимаются цвета CSS (рис. 9.16). Если вы установили Opera 11, не забудьте протестировать эту новую возможность, пройдя по следующей ссылке: **http://diveintohtml5.org/examples/input-type-color.html**.

Я благодарен Патрику Локу (Patrick Lauke) и Крису Миллзу (Chris Mills) за это дополнение к моей книге и предоставленную иллюстрацию. Рекомендую для прочтения следующую их статью: **http://dev.opera.com/articles/view/new-form-features-in-html5/**.

Валидизация формы

В этой главе я говорил о новых типах ввода и новых возможностях, например, об автоматической передаче фокуса ввода полям Web-форм и так далее, но до сих пор еще не упоминал о наиболее интересной возможности форм HTML5: об автоматической проверке правильности ввода. Рассмотрим общую проблему, типичную для ввода адресов электронной почты в поля Web-форм. Возможно, вы уже осуществляли некоторые операции по валидизации ввода на стороне клиента с помощью JavaScript, за которыми следовала валидизация на стороне сервера с помощью PHP, Python или какого-либо еще серверного языка командных сценариев. HTML5 никогда не сможет заменить проверку правильности ввода на стороне сервера, но вот валидизацию ввода на стороне клиента он когда-нибудь заменит.

Краткая сводка информации по поддержке валидизации Web-форм современными браузерами и мобильными устройствами приведена в табл. 9.5.

Таблица 9.5. *Поддержка валидизации Web-форм в современных браузерах и мобильных устройствах*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	4.0+	5.0+	6.0+	9.0+	.	.

С валидизацией адресов электронной почты на JavaScript существуют две большие проблемы:

- У поразительного количеств ваших посетителей (около 10%) не будет активизирована поддержка JavaScript;
- Вы реализуете ее неправильно.

Да, серьезно — вы реализуете ее неправильно. Определить, является ли случайная строка символов допустимым адресом электронной почты — это невероятно сложная задача (**http://www.regular-expressions.info/email.html**). Чем круче вы сами, как программист, тем сложнее она становится (**http://www.ex-parrot.com/pdw/Mail-RFC822-Address.html**). Я еще не убедил вас в том, что задача действительно очень сложна? Так посмотрите и убедитесь: **http://haacked.com/archive/2007/08/21/i-knew-how-to-validate-an-email-address-until-i.aspx**. Не лучше ли переложить всю эту работу на ваш браузер?

Хорошая новость: Opera валидизирует поля `type="email"` (рис. 9.17), причем эта возможность присутствует в браузере, начиная с Opera 9. Единственное, что необходимо добавить в разметку — указать тип ввода `<input type="email">`. Когда пользователь Opera попытается ввести в поле формы с типом `<input type="email">` адрес электронной почты, Opera автоматически предложит валидизацию адреса в соответствии с требованиями RFC, даже при блокированной поддержке скриптов.

HTML5 также предлагает и валидизацию Web-адресов, введенных в поля `<input type="url">`, и чисел, введенных в поля `<input type="number">` (рис. 9.18). При валидизации чисел в расчет берутся также атрибуты `min` и `max`, так что браузер не позволит вам передать в форму число, например, превышающее максимально допустимое значение.

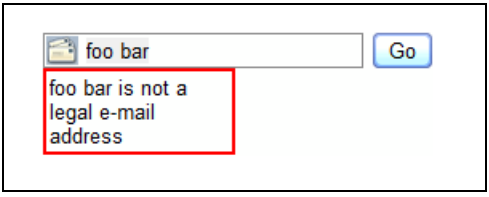


Рис. 9.17. Opera валидизирует поля `<input type="email">`

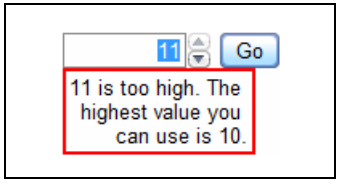


Рис. 9.18. Opera валидизирует поля `<input type="number">`

Чтобы активизировать валидизацию форм HTML5, специальной разметки не требуется; она активна по умолчанию. Чтобы отключить валидизацию форм, следует использовать атрибут novalidate, как показано в листинге 9.12.

Листинг 9.12. Отключение валидизации форм HTML5

```
<form novalidate>
  <input type="email" id="addr">
  <input type="submit" value="Subscribe">
</form>
```

Разработчики браузеров не особо спешат с реализацией поддержки проверки корректности пользовательского ввода в формы по стандарту HTML5. На настоящий момент объявлено, что эта поддержка будет реализована в Firefox 4

(**https://developer.mozilla.org/en/HTML/HTML5/Forms_in_HTML5#Constraint_Validation**). К сожалению, разработчики Safari и Chrome выпустили версии с неполной реализацией поддержки валидизации Web-форм, которая может "дать вам подножку": их браузеры валидизируют элементы управления форм, но не предоставляют никакой видимой обратной связи, когда введенные в форму данные не проходят проверки. Иными словами, если вы, например, введете некорректную (или неправильно отформатированную) дату в поле `type="date"`, то Safari и Chrome не выполнят отправки введенных данных, но и не сообщат вам, почему данные не были переданы. Вместо этого они установят фокус ввода на поле, которое содержит некорректное значение, но не выведут сообщения об ошибке, как это сделают Opera и Firefox 4.

Обязательные поля

Валидизация форм HTML5 не ограничивается проверкой типов данных в каждом поле. Вы можете также указать, что некоторые из полей являются обязательными для заполнения. Чтобы форма была передана, в обязательные поля (required fields) непременно должны быть введены значения.

Краткая сводка информации о поддержке обязательных для заполнения полей в современных браузерах и мобильных устройствах приведена в табл. 9.6.

Таблица 9.6. *Поддержка элемента <input required> в современных браузерах и мобильных устройствах*

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	4.0+	.	.	9.0+	.	.

Пример разметки полей, обязательных для заполнения, приведен в листинге 9.13.

Листинг 9.13. Пример разметки полей Web-формы, обязательных для заполнения

```
<form>
  <input id="q" required>
  <input type="submit" value="Search">
</form>
```

Протестируйте элемент `<input required>` в своем браузере (**http://diveintohtml5.org/examples/input-required.html**). Вот как реализует поддержку обязательных полей Opera 11 (рис. 9.19).

Браузеры могут изменять вид, который обязательные для заполнения поля имеют по умолчанию. Например, вот как обязательное для заполнения поле выглядит в Mozilla Firefox 4.0 (рис. 9.20).

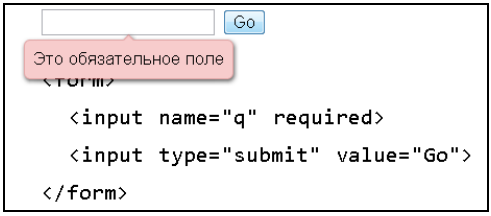


Рис. 9.19. В Opera поддержка элемента `<input required>` организована очень хорошо

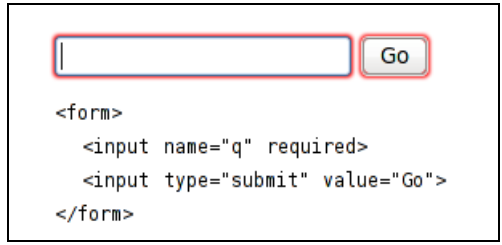


Рис. 9.20. В Firefox 4.0 вид полей, обязательных для заполнения, отличается от вида по умолчанию — такие поля обведены красной рамкой

Далее, если вы попытаетесь передать форму, не заполнив обязательное поле, Firefox 4.0, как и Opera, выведет всплывающую информационную панель, сообщающую вам о том, что это поле обязательно для заполнения и не может быть оставлено пустым.

Рекомендации по дальнейшему чтению

Спецификации и стандарты:

- О типах полей ввода — **http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#states-of-the-type-attribute**
- Об атрибуте `<input placeholder>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/common-input-element-attributes.html#the-placeholder-attribute**
- Об атрибуте `<input autofocus>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#autofocusing-a-form-control**
- Об атрибуте `<form novalidate>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#attr-fs-novalidate**
- Об атрибуте `<input required>` — **http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#attr-fs-novalidate**
- О формах HTML5 в Mozilla Firefox 4.0+ — **https://developer.mozilla.org/en/HTML/HTML5/Forms_in_HTML5**

Библиотеки JavaScript:

- `Modernizr`, библиотека для выявления проверки поддержки HTML5 — **http://www.modernizr.com/**

Полезные статьи:

- *"Forward Thinking Form Validation"* — **http://www.alistapart.com/articles/forward-thinking-form-validation/**
- *"New form features in Opera 11"* — **http://dev.opera.com/articles/view/new-form-features-in-html5/**

Глава 10

"Распределенные", "Расширяемость" и другие необычные слова

В состав HTML5 входят более 100 элементов (**http://simon.html5.org/html-elements**). Некоторые из них — чисто семантические (см. *главу 3*), тогда как другие являются просто контейнерами для командных сценариев, написанных с помощью API (см. *главу 4*). На протяжении всей истории HTML (см. *главу 1*), разработчики стандартов постоянно спорили о том, какие элементы должны быть включены в состав языка. Должен ли язык HTML включать элемент `<figure>`? А элемент `<person>`? Как насчет элемента `<rant>`? Решения принимаются, спецификации пишутся, авторы создают Web-страницы, исполнители занимаются реализацией, и "паутина" Web расползается все дальше и дальше.

Конечно, HTML не может удовлетворить любого. Ни один стандарт этого не может. Некоторые идеи просто "приходятся не ко двору". Например, в HTML5 нет никакого элемента `<person>`. Черт возьми, никакого элемента `<rant>` там тоже нет! И хотя никто и ничто не может помешать вам включить в Web-страницу элемент `<person>`, но он не будет признан корректным (**http://html5.validator.nu/**), не будет непротиворечиво восприниматься различными браузерами (**http://diveintohtml5.org/semantics.html#unknown-elements**) и может начать конфликтовать с будущими спецификациями HTML, если мы впоследствии захотим его добавить.

Ладно, раз добавление собственных элементов — это не ответ, тогда что должен делать Web-автор, склонный к семантически осмысленному использованию Web-элементов? Попытки расширения предыдущих версий HTML уже предпринимались. Наиболее популярный метод заключается в использовании микроформатов (microformats)[1], при котором в HTML 4 используются атрибуты `class` и `rel`. Еще один вариант — это опция RDFa (Resource Description Framework – in – attributes)[2], которая изначально разрабатывалась для использования в XHTML, но теперь переносится также и в HTML (**http://www.w3.org/TR/rdfa-in-html/**).

Микроформаты и RDFa имеют и свои сильные, и свои слабые стороны. Для достижения одной и той же цели они используют радикально различные подходы: расширение Web-страниц дополнительными семантическими значениями, не являющимися частью основы языка HTML. Я не собираюсь превращать эту главу в "священную войну" (для этого, определенно, понадобится ввести элемент `<rant>`).

---

[1] См. **http://microformats.org/**.

[2] См. **http://www.w3.org/TR/rdfa-syntax/**, **http://en.wikipedia.org/wiki/RDFa**.

Вместо этого я предложил бы сосредоточить внимание на третьей опции, которая тесно интегрируется с HTML5 и является частью самого этого стандарта: на микроданных (microdata).

Что такое микроданные?

Каждое слово в следующем предложении имеет значение, так что будьте внимательны.

> ***Замечание Профессора Разметки***
>
> Микроданные снабжают DOM аннотациями в виде пар "имя/значение" из определенного диапазона, взятых из индивидуальных словарей.

А теперь разберемся, что все это значит? Давайте начнем "с конца" и будем двигаться назад. Микроданные группируются вокруг индивидуальных словарей. Давайте будем рассматривать "набор всех элементов HTML5" как один словарь. Этот словарь включает в свой состав элементы, нужные для представления раздела (элемент `<section>`) или статьи (элемент `<article>`)[1], но не включает элементов, нужных, чтобы представить личность или событие. Если вы хотите представить кого-либо на Web-странице, вам потребуется определить ваш собственный словарь. Микроданные позволяют это сделать. Каждый может определить словарь микроданных и начать встраивать индивидуальные свойства на своих собственных Web-страницах.

Следующее, что нам необходимо узнать о микроданных, это то, что они работают с парами "имя/значение". Каждый словарь микроданных определяет набор именованных свойств. Например, словарь `Person` может определять такие свойства, как имя (`name`) и фотография (`photo`). Чтобы включить на свою Web-страницу конкретное свойство микроданных, вам потребуется в нужном месте указать имя нужного свойства. В зависимости от того, где вы объявляете имя свойства, микроданные руководствуются специфическими правилами по извлечению значения свойства. (Более подробно об этом будет рассказано в следующем разделе.)

Наряду с именованными свойствами, микроданные основываются на концепции определения диапазона ("scoping"). Самый простой путь составить себе представление о диапазонах микроданных заключается в том, чтобы думать о диапазонах микроданных, как о естественных отношениях "родитель-потомок" между элементами в DOM. Элемент `<html>` (**http://diveintohtml5.org/semantics.html#html-element**) обычно содержит двух потомков, `<head>` (**http://diveintohtml5.org/semantics.html#head-element**) и `<body>`. Элемент `<body>` обычно содержит множество дочерних элементов, каждый из которых может иметь собственные дочерние элементы. Например, ваша страница может содержать элемент `<h1>` в составе элемента `<hgroup>`, который в свою очередь входит в состав элемента `<header>` (**http://diveintohtml5.org/semantics.html#header-element**) в составе элемента `<body>`. Таблица данных может содержать элемент `<td>` в составе элемента `<tr>` внутри элемента `<table>` (в соста-

[1] См. **http://diveintohtml5.org/semantics.html#new-elements**

ве элемента <body>). Микроданные повторно используют иерархическую структуру DOM, предоставляя таким образом способ сообщить следующее: "все свойства в составе этого элемента берутся из этого словаря". Это позволяет вам использовать более одного словаря микроданных на одной и той же странице. Вы даже можете встраивать словари микроданных внутрь друг друга, и все это — за счет повторного использования структуры DOM. Далее в этой главе я продемонстрирую примеры вложенных словарей.

Теперь, когда мы уже затронули DOM, давайте разовьем эту тему дальше. *Микроданные связаны с добавлением дополнительного семантического смысла к данным, которые уже видны на вашей Web-странице*. Микроданные разрабатывались не с тем, чтобы создать самостоятельный формат данных. Этот формат является дополнительным к HTML. Как вы увидите в следующем разделе, микроданные лучше всего работают там, где уже корректно используется HTML, но при этом словарь HTML еще не в достаточной мере выразителен. Микроданные — отличное средство тонкой настройки семантики данных, уже помещенных в DOM. Если данные, над семантикой которых вы работаете, уже находятся в DOM, вам следует вернуться на шаг назад и переоценить свои данные, выяснив, являются ли они тем решением, которое вам необходимо.

Внимательно вчитайтесь — стало ли более понятным следующее предложение: "микроданные дают аннотацию DOM с распределенными по диапазонам парами “имя/значение” из индивидуальных словарей". Я надеюсь, что вы улавливаете этот смысл. Если так, давайте перейдем ближе к делу.

Модель данных "микроданные"

Определение собственного словаря микроданных —- дело несложное. Сначала вам потребуется пространство имен (namespace), которое представляет собой просто URL. Пространство имен URL может указывать просто на работающую Web-страницу, хотя это и не является обязательным требованием. Представим себе, что я хочу создать словарь микроданных, который описывает конкретную личность. Если мне принадлежит домен **data-vocabulary.org**, то в качестве моего словаря микроданных я могу воспользоваться следующим URL: **http://data-vocabulary.org/Person**. Это — простой способ создания глобально уникального идентификатора: выберите URL домена, который вы контролируете.

В этом словаре мне необходимо определить некоторые именованные свойства. Давайте начнем с трех основных свойств:

- `name` (ваше полное имя)
- `photo` (ссылка на вашу фотографию)
- `url` (ссылка на ассоциированный с вами Web-сайт — например, ваш блог или профиль Google)

Некоторые из свойств представляют собой URL, другие — обычный текст. Каждый из этих элементов подлежит естественной форме разметки еще прежде, чем вы начнете продумывать микроданные, словари, или что бы то ни было еще. Представьте себе, что у вас есть страница профиля или страница **About**. Возможно,

ваше имя помечено как заголовок, элементом <h1>. Возможно, ваша фотография указана элементом <img>, поскольку вы предполагаете, посетители будут ее просматривать. Возможно, что любые другие URL, ассоциированные с вашим профилем, оформлены как гиперссылки, чтобы посетители могли раскрывать их щелчком мыши. Далее, давайте для определенности предположим, что весь ваш профиль "обернут" элементом <section>, чтобы выделить его из остального содержимого страницы. Например, все это может выглядеть так, как показано в листинге 10.1.

Листинг 10.1. Пример страницы личного профиля "Обо мне"

```
<section>
  <h1>Mark Pilgrim</h1>
  <p><img src="http://www.example.com/photo.jpg" alt="[me smiling]"></p>
  <p><a href="http://diveintomark.org/">weblog</a></p>
</section>
```

Модель данных "микроданные" состоит из именованных пар ключей и значений. Имя свойства микроданных (например, `name`, `photo` или `url`, как в этом примере) всегда объявляется как элемент HTML. Значение, соответствующее свойству, извлекается из DOM этого элемента. Для большинства элементов HTML значение свойства представляет собой просто текстовое содержимое элемента. Но, тем не менее, существует и ряд исключений, вкратце перечисленных в табл. 10.1.

***Таблица 10.1.** Модель микроданных HTML5 — значения свойств микроданных и откуда они берутся*

Элемент	Значение
<meta>	Атрибут content
<audio>	
<embed>	
<iframe>	
<img>	
<source>	
<video>	Атрибут src
<a>	
<area>	
<link>	Атрибут href
<object>	Атрибут data
<time>	Атрибут datetime
Все остальные элементы	Текстовое содержимое

"Добавление микроданных" на вашу страницу — это вопрос добавления атрибутов к уже имеющимся элементам HTML. Первое, что вы делаете — вы декларируете, какой словарь микроданных вы используете, добавляя атрибут `itemtype`. Второе, что вам необходимо сделать — это определить диапазон словаря, используя атрибут `itemscope`. В рассматриваемом примере все данные, для которых мы хотим определить семантический смысл, находятся в элементе `<section>`, так что мы объявляем атрибуты `itemtype` и `itemscope` для элемента `<section>`.

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

Ваше имя — это первый фрагмент данных в составе элемента `<section>`. Он обрамлен элементом `<h1>`. Элемент `<h1>` не имеет никакой специальной обработки в модели микроданных HTML5, так что он подпадает под общее правило, действующее в отношении "всех остальных элементов", где значение свойства микроданных представляет собой обычный текстовый элемент. (Это правило будет работать точно так же, если вы "обернете" свое имя элементами `<p>`, `<div>` или `<span>`).

```
<h1 itemprop="name">Mark Pilgrim</h1>
```

Проще говоря, это правило гласит: "вот свойство `name` из словаря `http://data-vocabulary.org/Person`, и это свойство имеет значение `Mark Pilgrim`".

Далее следует свойство `photo`. Предполагается, что это — URL. В соответствии с моделью микроданных HTML5, "значение" элемента `<img>` представляет собой атрибут `src`. Посмотрите, URL вашей фотографии уже находится в атрибуте `<img src>`. Все, что вам требуется — это объявить, что элемент `<img>` представляет собой свойство `photo`, как показано в листинге 10.2.

Листинг 10.2. Определение свойства микроданных photo

```
<p><img itemprop="photo"
        src="http://www.example.com/photo.jpg"
        alt="[me smiling]"></p>
```

Проще говоря, этот код указывает нам на то, что это — свойство `photo` из словаря `http://data-vocabulary.org/Person`, и значение этого свойства — `http://www.example.com/photo.jpg`.

Наконец, свойство `url` тоже представляет собой URL. В соответствии с моделью микроданных HTML5, "значение" элемента `<a>` представляет собой его атрибут `href`. И здесь снова все очень хорошо укладывается в вашу существующую разметку. Все, что вам требуется проделать, сводится к тому, чтобы сказать, что ваш существующий элемент `<a>` представляет собой свойство `url`:

```
<a itemprop="url" href="http://diveintomark.org/">dive into mark</a>
```

Попросту говоря, этот код эквивалентен следующему высказыванию: "вот свойство `url` из словаря `http://data-vocabulary.org/Person`, и значение этого свойства — `http://diveintomark.org/`".

Конечно, если ваша разметка выглядит чуть иначе, то проблемы это не представляет. Вы можете добавлять свойства и значения микроданных к любой разметке

HTML, даже самой замысловатой, в стиле "таблиц-и бог-знает-чего-еще", какой был принят в XX веке. Хотя я и не рекомендую этот тип разметки, он все еще достаточно распространен, и вы можете добавлять к нему микроданные, как в примере, продемонстрированном в листинге 10.3.

Листинг 10.3. Замысловатый пример разметки, который лично я не рекомендую. Во имя бога, не делайте этого!

```
<TABLE>
  <TR><TD>Name<TD>Mark Pilgrim
  <TR><TD>Link<TD>
    <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
</TABLE>
```

Для разметки свойства `name` просто добавьте атрибут `itemprop` для ячейки таблицы, которая содержит имя. Ячейки таблицы не имеют специальных правил для таблицы значений свойств микроданных, поэтому они получают значение по умолчанию "свойство микроданных как текстовое содержимое".

```
  <TR><TD>Name<TD itemprop="name">Mark Pilgrim
```

Добавление свойства `url` выглядит более запутанным. Эта разметка не использует элемента `<a>` так, как это полагается правилами. Вместо того чтобы поместить цель, на которую указывает ссылка, в атрибут `href` не помещается ничего полезного, а вместо этого применяется JavaScript в атрибуте `onclick` для вызова функции (не показанной), которая извлекает URL и осуществляет переход к нему. Для "дополнительного бонуса" давайте притворимся, что функция также открывает ссылку в небольшом всплывающем окне без полос прокрутки. Разве работа в Internet не была сплошным удовольствием в прошлом веке?

В любом случае, вы все же можете преобразовать эту разметку в свойство микроданных, вам только нужно проявить чуть более креативного подхода. Непосредственное использование элемента `<a>` даже не обсуждается. Цель ссылки не является атрибутом `href`, и нет способа отменить правило, гласящее, что "в элементе `<a>` ищите свойство микроданных в атрибуте `href`". Но вы можете добавить элемент-"обертку" вокруг всей этой мешанины и воспользоваться им для добавления свойства микроданных `url`.

Результат преобразований показан в листинге 10.4.

Листинг 10.4. Результат преобразований для подавления разметки HTML

```
<TABLE itemscope itemtype="http://data-vocabulary.org/Person">
  <TR><TD>Name<TD>Mark Pilgrim
  <TR><TD>Link<TD>
    <span itemprop="url">
      <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
    </span>
</TABLE>
```

Так как элемент не имеет специальной обработки, он использует правило по умолчанию, гласящее, что "свойство микроданных представляет собой текстовое содержимое". "Текстовое содержимое" не означает "всю разметку внутри элемента" (как вы получили бы, используя, например, DOM-свойство innerHTML). Это означает "просто текст", не более того. В данном случае, это будет просто строка http://diveintomark.org/, текстовое содержимое элемента <a> внутри элемента , а не разметка HTML.

Подведем итоги: вы можете добавлять свойства микроданных к любой разметке. Если вы правильно применяете HTML, вы обнаружите, что вам будет проще добавлять микроданные, нежели в том случае, когда ваша разметка "хромает", но, тем не менее, сделать это можно всегда.

Разметка личных данных

Между прочим, наши начальные примеры, с которых мы начали эту главу, еще не полностью завершены. На самом деле существует специальный словарь микроданных, предназначенный именно для разметки информации о людях, и пользоваться им очень просто. Давайте рассмотрим его повнимательнее.

Простейший способ интегрировать микроданные в персональный Web-сайт предоставляется вашей страницей **About**. Конечно же, у вас есть такая страница, не так ли? Если нет, вы можете создать ее на основе образца страницы **About** (**http://diveintohtml5.org/examples/person.html**), дополнив ее расширенной семантикой. Окончательный результат находится здесь: **http://diveintohtml5.org/examples/person-plus-microdata.html**.

Давайте рассмотрим "сырую" разметку перед добавлением свойств микроданных, которая приведена в листинге 10.5.

Листинг 10.5. Пример "сырой" разметки персональной страницы профиля в блоге перед добавлением свойств микроданных

```
<section>
  <img width="204" height="250"
       src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
       alt="[Mark Pilgrim, circa 2000]">

  <h1>Contact Information</h1>
  <dl>
    <dt>Name</dt>
    <dd>Mark Pilgrim</dd>

    <dt>Position</dt>
    <dd>Developer advocate for Google, Inc.</dd>

    <dt>Mailing address</dt>
```

```
    <dd>
      100 Main Street<br>
      Anytown, PA 19999<br>
      USA
    </dd>
  </dl>
  <h1>My Digital Footprints</h1>
  <ul>
    <li><a href="http://diveintomark.org/">weblog</a></li>
    <li><a href="http://www.google.com/profiles/pilgrim">Google
profile</a></li>
    <li><a href="http://www.reddit.com/user/MarkPilgrim">Reddit.com
profile</a></li>
    <li><a href="http://www.twitter.com/diveintomark">Twitter</a></li>
  </ul>
</section>
```

Первое, что вам необходимо сделать в любом случае — объявить используемый вами словарь и диапазон свойств, которые вы собираетесь добавить. Это делается с помощью атрибутов `itemtype` и `itemscope` для внешнего элемента, который заключает в себя фактические данные. В данном случае, это будет элемент `<section>`.

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

Сравнивайте изменения в обоих файлах! Исходный файл: **person.html**, после внесения изменений: файл **person-plus-microdata.html**.

Теперь вы можете начинать определять свойства микроданных из словаря **http://data-vocabulary.org/Person**. Но где находятся эти свойства? Список их вы сможете увидеть, открыв в своем браузере адрес **http://www.data-vocabulary.org/Person/**. Спецификация микроданных этого не требует, но я бы точно и с уверенностью сказал, что это — один из лучших подходов. В конце концов, если вы хотите, чтобы разработчики действительно применяли ваш словарь микроданных на практике, вам следует его хорошо документировать. А куда лучше всего поместить эту документацию, как не в URL самого словаря? Список свойств микроданных из словаря **http://data-vocabulary.org/Person** приведен в табл. 10.2.

Таблица 10.2.** Список свойств микроданных из словаря **http://data-vocabulary.org/Person

Свойство	Описание
`name`	Имя
`nickname`	"Ник"
`photo`	Ссылка на изображение
`title`	Титул или название должности (например, "Финансовый директор")
`role`	Роль, которую человек выполняет (например, "бухгалтер")

Таблица 10.2 *(окончание)*

Свойство	Описание
`url`	Ссылка на Web-страницу (например, на домашнюю страницу этого человека)
`affiliation`	Отношение, которое данный человек имеет к ассоциированной с ним организации (например, работодатель)
`friend`	Указывает на дружбу или другие социальные отношения с другим человеком
`contact`	Идентифицирует социальные связи между этим человеком и другим лицом
`acquaintance`	Идентифицирует социальные связи между этим человеком и другим лицом
`address`	Адрес данного человека. Может иметь такие подсвойства, как `street-address`, `locality`, `region`, `postal-code` и `country-name`

Первое, что привлекает внимание в этом образце моей личной страницы **About** — это моя фотография. Естественно, она помечена тегом `<img>`. Чтобы объявить, что данный элемент `<img>` — это фотография из моего профиля, все, что необходимо сделать, сводится к добавлению атрибута `itemprop="photo"` к элементу `<img>`, как показано в листинге 10.6.

Листинг 10.6. Изменение, внесенное в исходный код страницы с тем, чтобы объявить о том, что данный элемент `<img>` представляет собой фотографию из моего профиля

```
<img itemprop="photo" width="204" height="250"
    src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
    alt="[Mark Pilgrim, circa 2000]">
```

Следите за изменениями в обоих файлах! Исходный файл: **person.html**, после внесения изменений: файл **person-plus-microdata.html**.

Где находится значение свойства микроданых? Здесь же, в атрибуте `src`. Если вы вспомните из модели микроданных HTML5, то "значение" элемента `<img>` представляет собой его атрибут `src`. Каждый элемент `<img>` имеет атрибут `src` — в противном случае он оказался бы просто "битой" картинкой — а `src` всегда представляет собой URL. Видите? Если вы корректно пользуетесь HTML, работать с микроданными будет очень просто!

Далее, этот элемент `<img>` не является единственным на странице. Это — дочерний элемент элемента `<section>`, того самого, который мы только что объявили с помощью атрибута `itemscope`. Микроданные повторно используют существующие связи "родитель-потомок" между элементами страницы с тем, чтобы определить отношения диапазонов (scoping) свойств микроданных. Иными словами, я пытаюсь сказать следующее: "данный элемент `<section>` представляет конкретного

человека. Любые свойства микроданных, которые вы можете обнаружить в наличии у потомков этого элемента <section>, представляют собой свойства, принадлежащие данному человеку". Если это вам помогает, можно думать об элементе <section> как о задающем тему некоторого предложения. Атрибут itemprop представляет собой сказуемое этого предложения, например, что-то наподобие "представлен на картинке". Значение свойства микроданных (microdata property value) представляет собой подлежащее в этом предложении. Выглядеть это будет так:

Этот человек [явно указанный, значение берется из <section itemscope itemtype="...">]

изображен на фотографии [явно указанной, значение берется из <img itemprop="photo">]

http://diveintohtml5.org/examples/2000_05_mark.jpg [неявно указанный, из атрибута <img src>]

Тему можно определить только один раз, путем включения атрибутов itemscope и itemtype во внешний элемент <section>. "Сказуемое" определяется за счет того, что вы помещаете атрибут itemprop="photo" в элемент <img>. Подлежащее в предложении вообще не нуждается в особой разметке, потому что модель микроданных HTML5 гласит, что значение свойства элемента <img> представляет собой его атрибут src.

Переходя к следующему элементу разметки, представленному в листинге 10.7, мы видим заголовок <h1> и начало списка <dl>. Ни <h1>, ни <dl> не нуждаются в разметке микроданными. Не каждый элемент разметки HTML должен представлять собой свойство микроданных. Микроданные предоставляют информацию о самих свойствах, а не о разметке или заголовках, которые их окружают. Этот элемент <h1> не является свойством; он представляет собой просто заголовок. Аналогично, элемент <dt>, гласящий Name, тоже не является свойством, это — просто метка.

Листинг 10.7. Следующий элемент разметки

```
<h1>Contact Information</h1>
<dl>
  <dt>Name</dt>
  <dd>Mark Pilgrim</dd>
```

Так где же находится настоящая информация? Она располагается в элементе <dd>, так что именно туда нам необходимо поместить атрибут itemprop. Что это за свойство? Это свойство name. Где находится значение этого свойства? В тексте, окруженном элементом <dd>. Нуждается ли оно в разметке? Модель микроданных HTML5 (см. табл. 10.1) утверждает, что нет, поскольку элементы <dd> не имеют специальной обработки, так что значение свойства представляет собой просто текст, находящийся внутри элемента.

В данном рассматриваемом примере это — просто мое имя (Mark Pilgrim)

```
<dd itemprop="name">Mark Pilgrim</dd>
```

Следите за изменениями в обоих файлах! Исходный файл: **person.html**, после внесения изменений: файл **person-plus-microdata.html**.

Что мы хотели всем этим сказать? А вот что: "Этого человека зовут `Mark Pilgrim`". Вот и все. А теперь перейдем к дальнейшему обсуждению.

Следующие два свойства несколько сложнее. Разметка, предшествующая добавлению микроданных, приведена в листинге 10.8.

Листинг 10.8. Разметка свойств на странице профиля, предшествующая добавлению микроданных

```
<dt>Position</dt>
<dd>Developer advocate for Google, Inc.</dd>
```

Если вы посмотрите на определение словаря `Person`, то текст `"Developer advocate for Google, Inc."` на самом деле охватывает два свойства: `title` (`"Developer advocate"`) и `affiliation` (`"Google, Inc."`). Как это можно выразить в микроданных? Краткий ответ будет таким: этого сделать нельзя. Микроданные не предоставляют способа разбиения фрагментов текста на отдельные свойства. Это значит, что вы не можете сказать "первые 18 символов этого текста представляют одно свойство микроданных, а последние 12 — это уже другое свойство микроданных".

Но не расстраивайтесь, еще не все потеряно. Представьте себе, что вы хотите отформатировать текст `"Developer advocate"`, оформив его другим шрифтом, отличающимся от шрифта, которым оформлен текст `"Google, Inc."`. Сделать это с помощью CSS тоже нельзя. И что вы предпримете? Сначала вы "обернете" символы "фиктивными" элементами, наподобие `<span>`, а затем примените разные правила CSS к каждому из этих элементов `<span>`.

В отношении микроданных применяется тот же самый подход. В данном случае у нас есть два различных информационных фрагмента: `title` и `affiliation`. Если вы заключите каждый элемент в фиктивный элемент `<span>`, вы сможете объявить каждый из этих элементов `<span>` как отдельное свойство микроданных. Данный подход продемонстрирован в листинге 10.9.

**Листинг 10.9. Оформление свойств микроданных с помощью фиктивных элементов **

```
<dt>Position</dt>
<dd><span itemprop="title">Developer advocate</span> for
    <span itemprop="affiliation">Google, Inc.<span></dd>
```

Следите за изменениями в обоих файлах! Исходный файл: **person.html**, после внесения изменений: файл **person-plus-microdata.html**.

Та-да-дамс! "Название должности: 'Developer advocate.' Этот человек является служащим компании Google, Inc.". Два предложения, два свойства микроданных. Совсем немного дополнительной разметки, но выигрыш того стоит.

Тот же самый подход будет полезен и при разметке адресов физического местопребывания. Словарь `Person` определяет свойство `address`, которое само является элементом микроданных. А это значит, что адрес и сам имеет собственный словарь (**http://data-vocabulary.org/Address**) и определяет собственные свойства. Словарь `Address` определяет 5 свойств: `street-address`, `locality`, `region`, `postal-code` и `country-name`.

Если вы программист, то вы, скорее всего, знакомы с "точечной" нотацией (dot notation), которая используется для определения объектов и их свойств. Задумайтесь об отношениях, приведенных в листинге 10.10.

Листинг 10.10. Использование "точечной нотации" для определения объектов и их свойств применительно к словарю Person

```
· Person
· Person.address
· Person.address.street-address
· Person.address.locality
· Person.address.region
· Person.address.postal-code
· Person.address.country-name
```

В этом примере весь адрес физического местоположения заключен в единственном элементе `<dd>`. (Здесь, как и ранее, элемент `<dt>` представляет собой просто метку, так что он не играет никакой роли в добавлении семантики к микроданным.) Обозначение адреса само по себе достаточно просто. Вам достаточно добавить атрибут `itemprop` к элементу `<dd>`, как показано в листинге 10.11.

Листинг 10.11. Разметка адреса физического местонахождения

```
<dt>Mailing address</dt>
<dd itemprop="address">
```

Следите за изменениями в обоих файлах! Исходный файл: **person.html**, после внесения изменений: файл **person-plus-microdata.html**.

Но помните, что свойство `address` само является элементом микроданных. Это значит, что нам необходимо добавить также атрибуты `itemscope` и `itemtype`, как показано в листинге 10.12.

Листинг 10.12. Добавление атрибутов `itemscope` и `itemtype` необходимо, потому что свойство address само является элементом микроданных

```
<dt>Mailing address</dt>
<dd itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
```

Следите за изменениями в обоих файлах! Исходный файл: **person.html**, после внесения изменений: файл **person-plus-microdata.html**.

Все это мы видели и раньше, но только для элементов верхнего уровня. Элемент `<section>` определяет `itemtype` и `itemscope`, и все элементы внутри элемента `<section>`, которые определяют свойства микроданных, собраны в пределах диапазона конкретного словаря. Но сейчас мы впервые встречаем "вложенные" диапазоны — определяем новые атрибуты `itemtype` и `itemscope` (для элемента `<dd>`) внутри уже существующего элемента (элемент `<section>`). Вложенный диапазон работает точно так же, как HTML DOM. Элемент `<dd>` имеет определенное количество дочерних элементов, каждый из которых ограничен диапазоном словаря, определенного для элемента `<dd>`. Как только элемент `<dd>` будет завершен закрывающим тегом `</dd>`, диапазон вернется к словарю, определенному в родительском элементе (в данном случае, `<section>`).

Свойства `address` страдают от той же самой проблемы, которую мы уже встречали в свойствах `title` и `affiliation`. Мы имеем всего лишь один длинный фрагмент текста, который нам нужно разбить на пять отдельных свойств микроданных. Решение будет точно таким же: "обернуть" каждый отдельный фрагмент информации в фиктивный элемент `<span>`, а затем объявить свойства микроданных для каждого элемента `<span>`, как показано в листинге 10.13.

Листинг 10.13. Разметка свойства `address` с помощью фиктивных элементов `<span>`

```
    <dd itemprop="address" itemscope
        itemtype="http://data-vocabulary.org/Address">
      <span itemprop="street-address">100 Main Street</span><br>
      <span itemprop="locality">Anytown</span>,
      <span itemprop="region">PA</span>
      <span itemprop="postal-code">19999</span>
      <span itemprop="country-name">USA</span>
    </dd>
  </dl>
```

Следите за изменениями в обоих файлах! Исходный файл: **person.html**, после внесения изменений: файл **person-plus-microdata.html**.

Говоря простыми словами: "У этого человека есть адрес для отправки почтовой корреспонденции. Адрес по улице — '100 Main Street.' Адрес локального почтового отделения — 'Anytown.' Регион — 'PA.' Почтовый индекс — '19999.' Страна — США". Видите, все очень просто.

Вопрос к Профессору Разметки

В: Этот адрес для отправки почтовой корреспонденции специфичен для США?

О: Нет. Свойства словаря `Address` являются типовыми и могут описывать почтовый адрес для доставки бумажной корреспонденции в любой стране мира. Не в любом адресе будут присутствовать поля, описываемые каждым из свойств, но в этом нет

ничего страшного. Для некоторых адресов может потребоваться более одной "строки" для каждого свойства, но это тоже нормально. Например, если физический адрес местоположения включает номер дома и квартиры, то оба эти информационных фрагмента будут описываться одним подсвойством `street-address`, как показано в листинге 10.14.

Листинг 10.14. Использование типовых свойств словаря `Address`

```
<p itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">
    100 Main Street
    Suite 415
  </span>
  ...
</p>
```

Есть еще один аспект, касающийся этого примера страницы **About**": список URL. Словарь `Person` имеет для этого свойство, которое так и называется: `url`. На самом деле свойство `url` может быть чем угодно. (Ладно, оно должно представлять собой URL, но вы, вероятно, и сами об этом догадались.) Что я имею в виду, так это то, что свойство `url` определено нестрого. Оно может представлять собой любой вид URL, который вы можете ассоциировать с конкретным человеком: это может быть блог, фотогалерея или любой другой адрес сайта, в том числе — Facebook или Twitter.

Еще один важный аспект, заслуживающий упоминания — это то, что один и тот же человек может иметь множество свойств `url`. Чисто технически, любое свойство может появляться более одного раза, но до сих пор мы этим пока не пользовались. Например, у вас может быть два свойства `photo`, каждое из которых может указывать на свое изображение, имеющее собственный URL. В данном примере я хочу указать несколько разных URL: мой блог, мою страницу профиля Google, мой пользовательский профиль Reddit и мою учетную запись Twitter. В HTML это выглядит как список из ссылок: четыре элемента `<a>`, каждый из которых в составе собственного элемента `<li>`. В микроданных, каждый элемент `<a>` получает атрибут `itemprop="url"`, как показано в листинге 10.15.

Листинг 10.15. Разметка списка URL с помощью элементов микроданных

```
<h1>My Digital Footprints</h1>
<ul>
  <li><a href="http://diveintomark.org/"
         itemprop="url">weblog</a></li>
  <li><a href="http://www.google.com/profiles/pilgrim"
         itemprop="url">Google profile</a></li>
```

```
    <li><a href="http://www.reddit.com/user/MarkPilgrim"
           itemprop="url">Reddit.com profile</a></li>
    <li><a href="http://www.twitter.com/diveintomark"
           itemprop="url">Twitter</a></li>
  </ul>
```

В соответствии с моделью микроданных HTML5 (см. таблю 10.1), элементы `<a>` имеют специальную обработку. Значение свойства микроданных представляет собой атрибут `href`, а не текстовое содержимое дочернего элемента. Процессор микроданных фактически игнорирует текст каждой ссылки. Таким образом, если говорить простыми словами, это значит следующее: "Этот человек имеет ассоциированный URL **http://diveintomark.org/**. Кроме того, с ним ассоциирован еще один URL — **http://www.google.com/profiles/pilgrim**. Далее, он имеет ассоциированный URL **http://www.reddit.com/user/MarkPilgrim** и еще один ассоциированный URL — **http://www.twitter.com/diveintomark**".

Введение в Google Rich Snippets

Я хочу вернуться на шаг назад, на минуточку, и задаться вопросом: а зачем мы все это делаем? Мы добавляем новую семантику только как самоцель? Добавить, ради того, чтобы добавить? Не поймите меня неправильно — я очень люблю возиться с тегами, ровно так же, как и любой другой Web-разработчик. Но при чем тут микроданные? Зачем с ними возиться?

На самом деле, есть два класса приложений, пользующихся HTML и, в качестве расширения — микроданными HTML5:

- Web-браузеры
- Поисковые машины (search engines)

Для браузеров HTML5 определяет набор DOM API (**http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#microdata-dom-api**), предназначенных для извлечения элементов микроданных, свойств и значений с Web-страницы. На тот момент, когда я пишу эти строки, ни один браузер еще не поддерживает этих API. Ни один. Так что это своего рода тупик, по крайней мере, до тех пор, пока браузеры не преодолеют это отставание в своем развитии и не начнут поддерживать клиентский API.

Второй основной потребитель HTML — это поисковые машины. Что может поисковик сделать со свойствами микроданных, относящихся к конкретному человек? Представьте себе следующее: вместо того, чтобы просто отображать заголовок страницы и фрагмент текста, поисковик может интегрировать некоторую структурированную информацию и отобразить ее. Например, сюда могут относиться полное имя, название должности, работодатель, миниатюра страницы профиля или фото. Привлекло бы это ваше внимание? Мое бы привлекло.

Google поддерживает микроданные как часть своей программы Rich Snippets (**http://tinyurl.com/4zwouvy**). Когда Web-поисковик Google выполняет разбор вашей страницы и находит на ней структурированные свойства микроданных, кото-

рые соответствуют словарю **http://data-vocabulary.org/Person**, он выполняет разбор этих свойств и сохраняет их совместно с другими данными страницы. Google даже предоставляет удобный инструмент для просмотра того, как Google интерпретирует ваши свойства микроданных, которое называется Rich Snippets Testing Tool (**http://www.google.com/webmasters/tools/richsnippets**). Его тестирование с нашим примером страницы **About** с возможностями работы с микроданными дает вывод, представленный в листинге 10.16.

Листинг 10.16. Тестирование страницы About из моего блога с помощью Rich Snippets Testing Tool

```
Item
  Type: http://data-vocabulary.org/person
  photo = http://diveintohtml5.org/examples/2000_05_mark.jpg
  name = Mark Pilgrim
  title = Developer advocate
  affiliation = Google, Inc.
  address = Item( 1 )
  url = http://diveintomark.org/
  url = http://www.google.com/profiles/pilgrim
  url = http://www.reddit.com/user/MarkPilgrim
  url = http://www.twitter.com/diveintomark

Item 1
  Type: http://data-vocabulary.org/address
  street-address = 100 Main Street
  locality = Anytown
  region = PA
  postal-code = 19999
  country-name = USA
```

Вся информация как на ладони: свойство `photo` из атрибута `<img src>`, все четыре URL из списка атрибутов `<a href>`, и даже объект `address` (указанный как `Item 1`) и все его пять подсвойств.

А как Google использует эту информацию? В зависимости от обстоятельств. Жестких правил, регламентирующих отображение свойств микроданных, нет, иначе говоря, четко не определено, ни какие из них должны отображаться, ни как они должны отображаться, ни должны ли они отображаться вообще. Если кто-нибудь будет искать по ключевым словам `"Mark Pilgrim"`, и Google определит, что страница **About** заслуживает отображения в результатах, и при этом Google сочтет, что найденные на этой странице свойства микроданных достойны того, чтобы быть отображенными, то результаты поиска могут выглядеть примерно так, как показано на рис. 10.1.

Рис. 10.1. Примерный вид результатов поиска Google с использованием микроданных

Первая строка, **About Mark Pilgrim**, фактически является заголовком страницы, указанным в элементе `<title>`. Само по себе это не выглядит чем-то сверхъестественным; Google делает это для каждой страницы. Зато вторая строка целиком заполнена информацией, взятой непосредственно из аннотаций микроданных, которые мы добавили на страницу. **Anytown PA** — это часть адреса для рассылки почтовой корреспонденции, помеченной в словаре **http://data-vocabulary.org/Address**. **Developer advocate** и **Google, Inc.** — это два свойства из словаря **http://data-vocabulary.org/Person** (`title` и `affiliation`, соответственно).

Вот это уже действительно становится интересным. Вам нет необходимости быть крупной корпорацией, ведущей специальные переговоры с производителями поисковиков, чтобы добиться того, чтобы результаты поиска отображали информацию о вас так, как вам требуется. Вам достаточно затратить десять минут, чтобы аннотировать данные, которые вы размещаете о себе в Сети.

Вопрос к Профессору Разметки

В: Я проделал все, что вы рекомендовали, но результаты поиска, выданные Google, ничем не отличаются от того, что я получал раньше. Как добиться описываемых вами результатов?

О: "Google не гарантирует (**http://tinyurl.com/4zwouvy**), что разметка, используемая на конкретной странице конкретного сайта, будет использована при выдаче результатов поиска". Но, даже если Google примет решение и не пользоваться вашими аннотациями микроданных, какой-нибудь другой поисковик может ими и воспользоваться. Как и остальные компоненты HTML5, микроданные представляют собой открытый стандарт, за реализацию которого сможет взяться кто угодно. Таким образом, ваша задача — предоставить как можно большие объемы данных. И пусть остальные решают, как с ними обходиться. Возможно, что то, как они это сделают, вас даже удивит!

Разметка данных об организациях

Микроданные не ограничиваются единственным словарем. Страницы **About** предоставляют удобные возможности, но, практически наверняка, хотя бы одна такая страница у вас уже есть. Вам хочется большего? Давайте посмотрим, какие возможности предлагаются для предприятий и организаций.

Вот пример образца страницы с информацией о предприятии (**http://diveintohtml5.org/examples/organization.html**). Для начала давайте рассмотрим оригинальную разметку HTML, без использования микроданных. Она приведена в листинге 10.17.

Листинг 10.17. Пример страницы About с информацией о коммерческом предприятии, без использования микроданных

```
<article>
  <h1>Google, Inc.</h1>
  <p>
    1600 Amphitheatre Parkway<br>
    Mountain View, CA 94043<br>
    USA
  </p>
  <p>650-253-0000</p>
  <p><a href="http://www.google.com/">Google.com</a></p>
</article>
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

Кратко и очень просто. Вся информация об организации содержится внутри элемента `<article>`, поэтому давайте с него и начнем.

```
<article itemscope itemtype="http://data-vocabulary.org/Organization">
```

Как и в случае с разметкой информации о частных лицах, при работе с разметкой информации для организаций вам необходимо установить атрибуты `itemscope` и `itemtype` для внешнего элемента. В данном случае, внешним является элемент `<article>`. Атрибут `itemtype` объявляет используемый вами словарь (в данном случае — **http://data-vocabulary.org/Organization**), а атрибут `itemscope` декларирует, что все свойства, заданные вами для дочернего элемента, относятся к этому словарю.

А что у нас содержится в словаре `Organization`? Все очень просто и, если можно так выразиться, прямолинейно. На практике, часть того, что вы увидите здесь, уже должна быть вам знакома. Краткая сводка информации, содержащейся в словаре `Organization`, приведена в табл. 10.3.

***Таблица 10.3.** Свойства, содержащиеся в словаре `Organization`*

Свойство	Описание
`name`	Наименование предприятия или организации (например, "Initech")
`url`	Ссылка на домашнюю страницу этого предприятия или организации
`address`	Местоположение офиса этой организации. Здесь можно указать такие вспомогательные свойства, как `street-address`, `locality`, `region`, `postal-code` и `country-name`
`tel`	Телефонный номер предприятия или организации
`geo`	Географические координаты офиса организации (адреса, по которому фактически располагается ее офис или штаб-квартира). Это свойство всегда содержит два подсвойства, широту (`latitude`) и долготу (`longitude`)

Первый элемент разметки во внешнем элементе <article> — это <h1>. Данный элемент <h1> содержит наименование предприятия, так что мы поместим атрибут itemprop="name" непосредственно в элемент <h1>.

```
<h1 itemprop="name">Google, Inc.</h1>
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

В соответствии с моделью микроданных HTML5 (см. табл. 10.1), элементы <h1> не нуждаются в специальной обработке. Значение свойства микроданных представляет собой просто текстовое содержимое элемента <h1>. Говоря простыми словами, мы хотим сообщить, что "название организации — 'Google, Inc.'".

Далее следует адрес штаб-квартиры этой организации. Разметка адреса штаб-квартиры организации работает точно так же, как разметка адресов частных лиц для доставки бумажной корреспонденции. В первую очередь, добавьте атрибут itemprop="address" в состав внешнего элемента адреса (в рассматриваемом примере — элемент <p>). Это говорит о том, что данное свойство представляет собой адрес организации. А как насчет свойств самого адреса? Кроме того, мы должны определить атрибуты itemtype и itemscope, чтобы сообщить о том, что это — элемент address, который имеет собственные свойства. Эта разметка представлена в листинге 10.18.

Листинг 10.18. Добавление элементов разметки микроданных для разметки адреса штаб-квартиры предприятия или организации

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

Наконец, мы должны "обернуть" каждый отдельный фрагмент информации в фиктивный элемент , так чтобы добавить соответствующие имена свойств микроданных (street-address, locality, region, postal-code и country-name) к каждому из элементов . Эта разметка показана в листинге 10.19.

Листинг 10.19. Добавление имен свойств микроданных к разметке адреса предприятия или организации

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">1600 Amphitheatre Parkway</span><br>
  <span itemprop="locality">Mountain View</span>,
```

```
    <span itemprop="region">CA</span>
    <span itemprop="postal-code">94043</span><br>
    <span itemprop="country-name">USA</span>
  </p>
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

Проще говоря, этим мы сказали следующее: "У этой организации есть адрес. Часть строки, указывающая улицу, выглядит как `'1600 Amphitheatre Parkway'`. Свойство `locality` имеет значение `'Mountain View'`. Регион — `'CA'`. Почтовый индекс — `'94043'`. Название страны - `'USA'`".

Далее нужно указать телефонный номер организации. Телефонные номера особенно сложны для указания, и их точный синтаксис зависит от страны. Причем если вы звоните в другую страну, то ситуация осложняется. В данном примере мы указываем телефонный номер в США в формате, пригодном для того, чтобы делать звонок из любой точки в пределах той же страны.

```
<p itemprop="tel">650-253-0000</p>
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

Да, кстати, на тот случай если вы этого не заметили — словарь `address` вышел за пределы диапазона при закрытии его элемента `<p>`. Теперь мы снова вернулись к определению свойств словаря `Organization`.

Если вы хотите перечислить несколько телефонных номеров — может быть, один — для клиентов из США, и другой — для международных клиентов, то вы можете это сделать. Любое свойство микроданных можно повторить. Не забудьте только убедиться в том, что каждый телефонный номер содержится в собственном HTML-элементе, отдельно от метки, которую вы ему присвоили. Пример разметки с добавлением дополнительных телефонных номеров представлен в листинге 10.20.

Листинг 10.20. Пример разметки сведений об организации или предприятии с добавлением нескольких телефонных номеров

```
<p>
  US customers: <span itemprop="tel">650-253-0000</span><br>
  UK customers: <span itemprop="tel">00 + 1* + 6502530000</span>
</p>
```

В соответствии с моделью микроданных HTML5 (см. табл. 10.1), ни элемент `<p>`, ни элемент `<span>` не имеют специальной обработки. Значение свойства микроданных `tel` — это просто текстовое содержимое. Словарь микроданных `Organization`

не делает даже попыток определить различные части телефонного номера. Все свойство `tel` — это просто текст в свободной форме. Если вы хотите заключить в скобки код области или использовать пробелы вместо дефисов для разделения групп цифр, то вы можете это сделать. Если клиент, пользующийся микроданными, желает выполнять разбор телефонного номера, такие клиенты должны выполнять это самостоятельно.

Далее, мы имеем еще одно знакомое свойство: `url`. Точно так же, как вы ассоциируете URL с частными лицами, вы можете ассоциировать их и с организациями. Это может быть домашняя страница компании, страница с контактной информацией или какая угодно другая страница. Если это — URL страницы **About** или страницы, принадлежащей этой организации, пометьте ее атрибутом `itemprop="url"`, как показано в листинге 10.21.

Листинг 10.21. Пример ассоциирования URL с организацией

```
<p><a itemprop="url" href="http://www.google.com/">Google.com</a></p>
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

В соответствии с моделью микроданных HTML5 (см. табл. 10.1), элемент `<a>` имеет специальную обработку. Значение свойства микроданных — это значение атрибута `href`, а не текст ссылки. Говоря простыми словами, эта разметка означает следующее: "данная организация ассоциирована с URL **http://www.google.com/**". Ничего более конкретного об этой ассоциации не говорится, и текст "Google.com" сюда не включается.

Наконец, я хочу напоследок поговорить и о геопозиционировании (geolocation). Нет, не о W3C Geolocation API (см. *главу 6*). Здесь говорится об описании физического местоположения организации с помощью микроданных.

До последнего момента, все наши примеры концентрировались на разметке видимых данных. Я имею в виду, что если у вас есть элемент `<h1>` с названием компании, то вы добавляете атрибут `itemprop` к элементу `<h1>`, чтобы объявить, что видимый текст в заголовке фактически представляет собой имя организации. Или предположим, что у вас есть элемент `<img>`, который указывает на фотографию, так что вы добавляете атрибут `itemprop` к элементу `<img>`, чтобы объявить, что (видимое) изображение представляет собой фотографию данного лица.

В рассматриваемом примере информация `geolocation` выглядит не так. Нет видимого текста, указывающего точные значения географических широты и долготы (до четырех десятичных позиций!) организации. Фактически, пример **http://diveintohtml5.org/examples/organization.html** (без микроданных) вообще не имеет информации геопозиционирования. Он имеет ссылку на Google Maps, но даже URL этой ссылки не содержит координат широты и долготы. Он содержит аналогичную информацию в формате, специфичном для Google. Но даже если бы мы

имели ссылку на гипотетический сервис, отображающий географические координаты на карте, принимающий значения широты и долготы в виде параметров URL, мы все равно не имели бы способа выделять микроданные как отдельные части URL. Вы не можете объявить, что первый параметр запроса URL представляет собой широту, второй — долготу, а остальные параметры запроса к делу отношения не имеют.

Чтобы обрабатывать такие пограничные ситуации, как эта, HTML5 предоставляет способ аннотировать невидимые данные (invisible data). Данная методика должна применяться только как последнее средство. Если существует способ отобразить или визуализировать данные, о которых вы можете позаботиться, вам необходимо это сделать. Невидимые данные, прочесть которые может только машина, имеют тенденцию к быстрому устареванию. Это значит, что впоследствии кто-нибудь может увидеть данные и обновить видимый текст, но забыть сделать то же самое для невидимых данных. Такие вещи происходят чаще, чем вы можете подумать, и вы тоже обязательно попадете в такую ситуацию.

Тем не менее, бывают случаи, когда применение невидимых данных неизбежно. Может быть, ваш руководитель действительно хочет, чтобы информация геопозиционирования в формате, воспринимаемом машиной, была доступна, но при этом не хочет, чтобы интерфейс загромождался шестизначными числами, неудобными для восприятия. В этом случае невидимые данные — это единственно возможный вариант. Единственный выигрышный момент здесь — то, что вы можете поместить невидимые данные сразу же за видимым текстом, который их описывает, что может помочь напомнить тем, кто впоследствии будет обновлять видимый текст о том, что им необходимо обновить и невидимые данные, следующие сразу же за ним.

В рассматриваемом примере мы можем создать фиктивный элемент `<span>` внутри того же самого элемента `<article>`, как и остальные свойства объекта `Organization`, затем поместить невидимые данные геопозиционирования внутри элемента `<span>`, как показано в листинге 10.22.

Листинг 10.22. Пример добавления невидимых данных

```
  <span itemprop="geo" itemscope
    itemtype="http://data-vocabulary.org/Geo">
    <meta itemprop="latitude" content="37.4149" />
    <meta itemprop="longitude" content="-122.078" />
  </span>
</article>
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

Информация геопозиционирования определяется внутри собственного словаря, так же, как и адрес частного лица или организации. Следовательно, этот элемент `<span>` нуждается в трех атрибутах:

- `itemprop="geo"` сообщает о том, что данный элемент представляет свойство `geo` данной организации;
- `itemtype="http://data-vocabulary.org/Geo"` сообщает о том, какому словарю микроданных соответствуют свойства этого элемента;
- `itemscope` сообщает о том, что данный элемент представляет собой заключающий элемент для элемента микроданных с его собственным словарем (указанным в атрибуте `itemtype`). Все свойства внутри этого элемента представляют собой свойства **http://data-vocabulary.org/Geo**, а не заключающего **http://data-vocabulary.org/Organization**.

Следующий крупный вопрос, на который отвечает данный пример, звучит так: "Как вы собираетесь аннотировать невидимые данные?" Вы используете элемент `<meta>`. В предшествующих версиях HTML, вы могли пользоваться элементом `<meta>` только внутри элемента `<head>` на вашей странице (см. *главу 3*). В HTML5, вы можете использовать элемент `<meta>` где угодно. Именно это мы и проделали здесь.

```
<meta itemprop="latitude" content="37.4149" />
```

Следите внимательно за различиями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/organization.html**, файл после добавления микроданных: **http://diveintohtml5.org/examples/organization-plus-microdata.html**.

В соответствии с моделью микроданных HTML5 (см. табл. 10.1), элемент `<meta>` имеет специальную обработку. Значение свойства микроданных — это атрибут содержимого. Так как этот атрибут никогда не отображается явным образом, мы имеем идеальные предпосылки для работы с неограниченным количеством невидимых данных. Но с большими возможностями, которые мы получаем, мы должны нести и не меньшую ответственность. В данном случае, на вас ложится ответственность за то, чтобы синхронизировать ваши невидимые данные с окружающим их видимым текстом.

В Google Rich Snippets нет непосредственной поддержки для словаря `Organization`, поэтому у меня нет достаточно простых листингов результатов поиска, чтобы продемонстрировать их вам. Но организации серьезно полагаются на следующие два примера: события (`events`) и рецензии (`reviews`), а они как раз поддерживаются в Google Rich Snippets.

Разметка событий

Гадости случаются. Некоторые особенные гадости случаются в предопределенные моменты времени. Разве плохо будет, если вы сможете собщить поисковикам о том, когда именно ожидается конкретная пакость? Для этого даже есть своя "угловая скобка".

Давайте начнем рассмотрение вопроса с изучения вполне конкретного расписания моих выступлений с презентациями (**http://diveintohtml5.org/examples/event.html**). Рассмотрим пример, приведенный в листинге 10.23.

Листинг 10.23. Пример листинга с описанием расписаний моих выступлений

```
<article>
  <h1>Google Developer Day 2009</h1>
  <img width="300" height="200"
       src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
       alt="[Mark Pilgrim at podium]">
  <p>
    Google Developer Days are a chance to learn about Google
    developer products from the engineers who built them. This
    one-day conference includes seminars and “office hours”
    on web technologies like Google Maps, OpenSocial, Android,
    AJAX APIs, Chrome, and Google Web Toolkit.
  </p>
  <p>
    <time datetime="2009-11-06T08:30+01:00">2009 November 6, 8:30</time>
      &ndash;
    <time datetime="2009-11-06T20:30+01:00">20:30</time>
  </p>
  <p>
    Congress Center<br>
    5th kvetna 65<br>
    140 21 Praha 4<br>
    Czech Republic
  </p>
  <p><a
href="http://code.google.com/intl/cs/events/developerday/2009/home.html">GDD/P
rague home page</a></p>
</article>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Вся информация о конкретном мероприятии содержится внутри элемента `<article>`, поэтому именно туда нам и следует поместить атрибуты `itemtype` и `itemscope`.

```
<article itemscope itemtype="http://data-vocabulary.org/Event">
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

URL для словаря `Event` выглядит так: **http://data-vocabulary.org/Event**, причем в его составе содержится симпатичная маленькая диаграмма, описывающая свойства словаря. И каковы эти свойства? Они описаны в табл. 10.4.

Таблица 10.4. *Свойства, содержащиеся в составе словаря* `Event`

Свойство	Описание
`summary`	Наименование мероприятия
`url`	Ссылка на страницу с описанием мероприятия
`location`	Место проведения мероприятия. При желании может быть представлено вложенным описанием `Organization` или `Address`
`description`	Описание мероприятия
`startDate`	Дата начала мероприятия в формате по стандарту ISO (**http://www.iso.org/iso/date_and_time_format**)
`endDate`	Дата и время окончания мероприятия в формате ISO
`duration`	Продолжительность мероприятия в формате времени по стандарту ISO
`eventType`	Категория мероприятия (например,“Concert” или “Lecture”). Это — строка свободной формы, а не перечисляемый атрибут
`geo`	Географические координаты места проведения. Всегда содержит два подсвойства, широту (`latitude`) и долготу (`longitude`)
`photo`	Ссылка на фотографию или изображение, ассоциированные с мероприятием

Название мероприятия представляет собой элемент `<h1>`. В соответствии с моделью микроданных HTML5 (см. табл. 10.1), элементы `<h1>` не имеют специальной обработки. Значение свойства микроданных — это просто текстовое содержимое элемента `<h1>`. Все, что нам необходимо сделать, сводится к добавлению атрибута `itemprop`, чтобы объявить, что данный элемент `<h1>` содержит название мероприятия.

```
<h1 itemprop="summary">Google Developer Day 2009</h1>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

В простой формулировке, это значит, что мы хотим сказать: "Это мероприятие называется `Google Developer Day 2009`".

Описание мероприятия содержит фотографию, которую можно снабдить свойством `photo`. Как можно было бы ожидать, фотография уже помечена элементом `<img>`. Как и в случае со свойством `photo` в словаре `Person`, фотография в словаре `Event` представляет собой URL. Так как модель микроданных HTML5 гласит, что значение свойства элемента `<img>` представляет собой его атрибут `src`, единственное, что нам остается — это добавить атрибут `itemprop` к элементу `<img>`, как показано в листинге 10.24.

Листинг 10.24. Добавление микроданных к разметке описания мероприятия, содержащего фотографию

```
<img itemprop="photo" width="300" height="200"
     src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
     alt="[Mark Pilgrim at podium]">
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Проще говоря, здесь мы имеем в виду следующее: "Фотография, ассоциированная с этим мероприятием, находится по адресу **http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg**."

Далее идет более длинное описание мероприятия, которое представляет собой абзац текста в свободной форме, который приведен в листинге 10.25.

Листинг 10.25. Добавление микроданных в описание мероприятия к свойству, представляющему собой текстовый абзац

```
<p itemprop="description">Google Developer Days are a chance to
learn about Google developer products from the engineers who built
them. This one-day conference includes seminars and "office
hours" on web technologies like Google Maps, OpenSocial,
Android, AJAX APIs, Chrome, and Google Web Toolkit.</p>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Следующий фрагмент информации представляет собой нечто новое. Как правило, мероприятия проводятся в заблаговременно назначенные дни, а начинаются и заканчиваются в заранее назначенное время. В HTML5 даты и времена должны быть размечены с помощью элемента `<time>`, и это мы уже проделывали. Таким образом, встает вопрос о том, как добавить свойства микроданных к этим элементам `<time>`? Вернувшись к модели микроданных HTML5 (см. табл. 10.1), мы обнаружим, что элемент `<time>` имеет специальную обработку. Значение свойства микроданных для элемента `<time>` представляет собой значение атрибута `datetime`. И посмотрите, свойства `startDate` и `endDate` словаря `Event` принимают значения в стиле даты ISO, точно так же, как и свойство `datetime` элемента `<time>`. Здесь, как и до сих пор, семантика базового словаря HTML отлично сходится с семантикой нашего индивидуально разрабатываемого словаря. Разметка дат начала и конца мероприятия с помощью микроданных сводится к следующему:

- Корректному использованию HTML (использованию элементов `<time>` для разметки информации о дате и времени).
- Добавлению единственного атрибута itemprop, как показано в листинге 10.26.

Листинг 10.26. Пример разметки дат начала и окончания мероприятия с помощью микроданных

```
  <p>
    <time itemprop="startDate" datetime="2009-11-06T08:30+01:00">2009 November
6, 8:30</time>
      &ndash;
    <time itemprop="endDate" datetime="2009-11-06T20:30+01:00">20:30</time>
  </p>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Проще говоря, мы сообщаем: "Данное мероприятие состоится 6 ноября 2009 года, в 8:30 утра и закончится 6 ноября 2009, в 20:30 (времена начала и конца указываются по местному времени Праги, GMT+1)".

Далее идет свойство `location`. Определение словаря `Event` (**http://www.data-vocabulary.org/Event/**) гласит, что это может быть либо свойство `Organization`, либо свойство `Address`. В данном случае мероприятие будет проводиться в помещении, специально для этого предназначенном — Пражском Дворце конгрессов. Разметив его как `Organization`, мы можем включить как наименование здания, так и его точный адрес.

Во-первых, давайте объявим, что элемент `<p>`, который содержит адрес, представляет собой свойство `location` мероприятия `Event`, и что данный элемент также является элементом микроданных, соответствующим словарю **http://data-vocabulary.org/Organization**. Это делается так, как показано в листинге 10.27.

Листинг 10.27. Разметка описания мероприятия с помощью микроданных из словаря http://data-vocabulary.org/Organization

```
<p itemprop="location" itemscope
        itemtype="http://data-vocabulary.org/Organization">
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Далее, разметим имя организации `Organization`, обернув имя фиктивным элементом `<span>` и добавив атрибут `itemprop` к элементу `<span>`.

```
    <span itemprop="name">Congress Center</span><br>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Вследствие правил диапазонов микроданных, `itemprop="name"` определяет свойство в словаре `Organization`, а не в словаре `Event`. Элемент `<p>`, определенный

в начале диапазона свойств `Organization`, и элемент `<p>` еще не закрыты завершающим тегом `</p>`. Любые свойства микроданных, которые мы здесь определяем, являются свойствами последнего использовавшегося словаря. Вложенные словари напоминают стек. Мы еще не выполняли "стаскивания" со стека, поэтому пока мы все еще говорим о свойствах `Organization`.

Фактически, мы собираемся добавить в стек третий словарь: `Address` для `Organization` для словаря `Event`. Эта разметка показана в листинге 10.28.

Листинг 10.28. Добавление третьего словаря

```
<span itemprop="address" itemscope
      itemtype="http://data-vocabulary.org/Address">
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Опять же, мы хотим разметить каждую часть адреса как отдельный элемент микроданных, поэтому нам снова нужны фиктивные элементы `<span>`, на которые мы "навесим" атрибуты `itemprop`. Если вы не успеваете за ходом моей мысли, вернитесь немного назад и перечитайте материалы о разметке адресов частных лиц и разметке адресов организаций. Разметка показана в листинге 10.29.

Листинг 10.29. Разметка каждой части адреса места проведения мероприятия как отдельного элемента микроданных

```
<span itemprop="street-address">5th kvetna 65</span><br>
<span itemprop="postal-code">140 21</span>
<span itemprop="locality">Praha 4</span><br>
<span itemprop="country-name">Czech Republic</span>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

`Address` больше не имеет других свойств, поэтому мы закрываем элемент `<span>`, который начинал диапазон `Address`, и выполняем "стаскивание" со стека.

```
</span>
```

Других свойств `Organization` тоже нет, поэтому мы закрываем элемент `<p>`, начинавший диапазон `Organization`, и еще раз выполняем "стаскивание" со стека.

```
</p>
```

Теперь мы вернулись к определению свойств `Event`. Следующим свойством является `geo`, представляющее точные географические координаты места проведения мероприятия `Event`. Оно использует тот же словарь `Geo`, который мы использовали для разметки географического местоположения организации в предыдущем разделе. Нам нужен элемент `<span>` для использования в качестве контейнера; он полу-

чает атрибуты itemtype и itemscope. Внутри элемента нам нужны два элемента <meta>, один — для свойства latitude, и один — для свойства longitude. Разметка этого свойства показана в листинге 10.30.

Листинг 10.30. Разметка свойства, описывающего точные географические координаты места проведения мероприятия

```
<span itemprop="geo" itemscope itemtype="http://data-vocabulary.org/Geo">
  <meta itemprop="latitude" content="50.047893" />
  <meta itemprop="longitude" content="14.4491" />
</span>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Мы закрыли элемент , который содержал свойства Geo, так что теперь мы вернулись к определению свойств Event. Последним среди них является свойство url, которое должно выглядеть уже знакомым. Ассоциирование URL с Event работает так же, как ассоциирование URL со словарями Person и Organization. Если вы корректно применяете HTML (размечая гиперссылки с помощью <a href>), то объявление того факта, что гиперссылка представляет собой свойство микроданных url, сводится к добавлению атрибута itemprop. Соответствующая разметка, демонстрирующая ассоциирование URL с конкретным мероприятием, представлена в листинге 10.31.

Листинг 10.31. Разметка, демонстрирующая ассоциирование URL с конкретным мероприятием

```
  <p>
    <a itemprop="url"

href="http://code.google.com/intl/cs/events/developerday/2009/home.html">
      GDD/Prague home page
    </a>
  </p>
</article>
```

Следите внимательно за изменениями в обоих файлах! Файл до добавления микроданных: **http://diveintohtml5.org/examples/event.html**, после добавления микроданных: **http://diveintohtml5.org/examples/event-plus-microdata.html**.

Наша страница с примером объявления о проводимом мероприятии сообщает и о втором мероприятии, а именно — о моем докладе на конференции по Кунг-фу в Монреале. Для краткости, я не буду здесь построчно рассматривать разметку этого события. В сущности, она такая же, как и та, что была выполнена для мероприя-

тия в Праге: элемент `Event` с встроенными элементами `Geo` и `Address`. Я просто упоминаю об этом, когда говорю о том, что одна страница может содержать множество анонсов разных мероприятий, каждое из которых снабжено разметкой с использованием микроданных.

Возвращаемся к Google Rich Snippets

В соответствии с Google Rich Snippets Testing Tool, эту информацию "пауки" Google извлекут из нашей страницы с примером сведений об анонсированных мероприятиях. Пример извлеченного содержимого показан в листинге 10.32.

Листинг 10.32. Пример информации, которую Google Rich Snippets Testing Tool может извлечь из нашей страницы с примером сведений об анонсированных мероприятиях

```
Item
    Type: http://data-vocabulary.org/Event
    summary = Google Developer Day 2009
    eventType = conference
    photo = http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg
    description = Google Developer Days are a chance to learn about Google
developer products from the engineers who built them. This one-day conference
includes seminars and office hours on web technologies like Goo...
    startDate = 2009-11-06T08:30+01:00
    endDate = 2009-11-06T20:30+01:00
    location = Item(__1)
    geo = Item(__3)
    url = http://code.google.com/intl/cs/events/developerday/2009/home.html
Item
    Id: __1
    Type: http://data-vocabulary.org/Organization
    name = Congress Center
    address = Item(__2)
Item
    Id: __2
    Type: http://data-vocabulary.org/Address
    street-address = 5th května 65
    postal-code = 140 21
    locality = Praha 4
    country-name = Czech Republic
Item
    Id: __3
    Type: http://data-vocabulary.org/Geo
    latitude = 50.047893
    longitude = 14.4491
```

Как видите, вся информация, добавленная нами в состав микроданных, здесь присутствует. Свойства представляют собой отдельные элементы микроданных, им присвоены внутренние идентификаторы (`Item__1`, `Item__2` и так далее). Это — не часть спецификации микроданных. Это просто соглашение, которое используется инструментарием тестировщика Google для линеаризации примеров вывода и демонстрации группировки встроенных элементов и их свойств.

На рис. 10.2 приведен пример того, как Google может представить эту информацию в результатах поиска. Опять же, я хочу предварить дальнейшее обсуждение оговоркой о том, что это — всего лишь пример. Google может изменить свой формат представления результатов поиска в любой момент, и то, что они вообще учтут вашу разметку микроданных, тоже не гарантируется. Мне очень жаль, если моя речь звучит, как заезженная пластинка, но я даже с юридической точки зрения обязан это сказать.

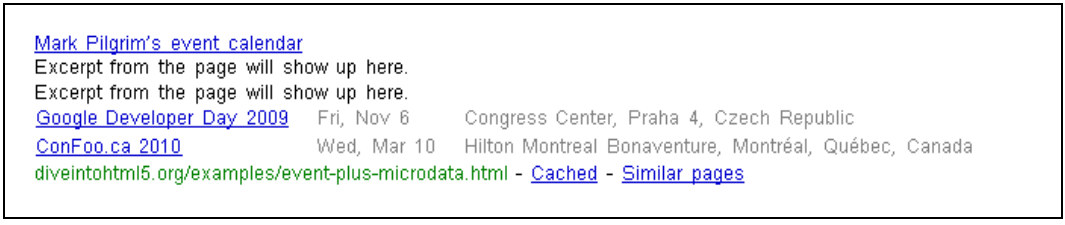


Рис. 10.2. Пример того, как Google может представить в результатах поиска нашу страницу с разметкой об анонсированных мероприятиях

После заголовка страницы и автоматически сгенерированного текста отрывка содержимого страницы, Google отображает (предположительно) разметку микроданных, которую мы добавили к странице, чтобы отобразить краткий список мероприятий. Обратите внимание на формат даты: `Fri, Nov 6`. Это не та строка, которая фигурировала где-то в нашей разметке HTML или микроданных. Мы использовали две полноформатных строки, отформатированных по стандарту ISO, `2009-11-06T08:30+01:00` и `2009-11-06T20:30+01:00`. Google принимает эти данные, определяет, что мероприятия имеют место в один и тот же день, и принимает решение отобразить только одну дату в формате, более удобном для восприятия человеком.

Теперь обратите внимание на физические адреса. Google предпочитает отобразить только наименование здания, страну и район, но не точный адрес. Это стало возможным за счет того, что мы разбили адрес на пять подсвойств — название, адрес, регион, местность и название страны — и разметили каждую часть адреса отдельным свойством микроданных. Google пользуется этим преимуществом и отображает сокращенный адрес. Другие потребители тех же микроданных могут сделать другой выбор и отображать их иначе. Здесь нет правильного или неправильного выбора. Ваша задача — предоставить как можно больше данных, и как можно точнее. Задача всех остальных — интерпретировать эти данные.

Разметка рецензий, обзоров и отзывов

Вот еще один пример разметки, позволяющий повысить удобство использования Web и, по возможности, удобство представления результатов поиска: бизнес-обзоры и рецензии на различную продукцию.

Вот, например, короткая рецензия, которую я написал на свою любимую пиццерию, расположенную недалеко от моего дома. Между прочим, этот ресторанчик действительно существует. Если вы как-нибудь окажетесь в тех краях, я вам его очень рекомендую. Давайте посмотрим на исходную разметку моей рецензии, которая представлена в листинге 10.33.

Листинг 10.33. Исходный текст разметки моей рецензии на мой любимый ресторан

```
<article>
  <h1>Anna's Pizzeria</h1>
  <p>????? (4 stars out of 5)</p>
  <p>New York-style pizza right in historic downtown Apex</p>
  <p>
    Food is top-notch. Atmosphere is just right for a 0neighborhood
    pizza joint.0 The restaurant itself is a bit cramped; if you're
    overweight, you may have difficulty getting in and out of your
    seat and navigating between other tables. Used to give free
    garlic knots when you sat down; now they give you plain bread
    and you have to pay for the good stuff. Overall, it's a winner.
  </p>
  <p>
    100 North Salem Street<br>
    Apex, NC 27502<br>
    USA
  </p>
  <p>- reviewed by Mark Pilgrim, last updated March 31, 2010</p>
</article>
```

Следите внимательно за изменениями в обоих файлах! Исходный вариант: **http://diveintohtml5.org/examples/review.html**, после добавления микроданных: **http://diveintohtml5.org/examples/review-plus-microdata.html**.

Эта рецензия содержится в элементе `<article>`, так что именно туда мы и поместим атрибуты `itemtype` и `itemscope`. Пространство имен URL для этого словаря — **http://data-vocabulary.org/Review**.

```
<article itemscope itemtype="http://data-vocabulary.org/Review">
```

Следите внимательно за изменениями в обоих файлах! Исходный вариант: **http://diveintohtml5.org/examples/review.html**, после добавления микроданных: **http://diveintohtml5.org/examples/review-plus-microdata.html**.

Какие свойства доступны в словаре `Review`? Хорошо, что вы об этом спросили. Эти свойства перечислены в табл. 10.5.

Таблица 10.5. *Краткое описание свойств, содержащихся в словаре микроданных* `Review`

Свойство	Описание
`itemreviewed`	Название объекта рецензии. Это может быть конкретный товар, продукт, услуга или предприятие
`rating`	Численное значение, выражающее собой оценку в баллах, выставленную оцениваемому объекту (обычно: от 1 до 5). Можно также использовать встроенный словарь **http://data-vocabulary.org/Rating** — как правило, это делается для оценки по нестандартной шкале
`reviewer`	Имя автора, написавшего рецензию
`dtreviewed`	Дата написания рецензии — в формате дат по стандарту ISO (**http://www.iso.org/iso/date_and_time_format**)
`summary`	Краткое резюме, обобщающее рецензию
`description`	Собственно текст самой рецензии

Первое свойство является очень простым: `itemreviewed` — это просто текст, и здесь он содержится в элементе `<h1>`, так что именно сюда мы и помещаем атрибут `itemprop`.

```
<h1 itemprop="itemreviewed">Anna's Pizzeria</h1>
```

Следите внимательно за изменениями в обоих файлах! Исходный вариант: **http://diveintohtml5.org/examples/review.html**, после добавления микроданных: **http://diveintohtml5.org/examples/review-plus-microdata.html**.

Я собираюсь пропустить фактический рейтинг и перейти сразу же к заключительной части. Следующие два свойства тоже очень просты. Свойство `summary` представляет собой просто краткое описание объекта вашей рецензии, а поле `description` должно содержать текст рецензии. Разметка этих свойств представлена в листинге 10.34.

Листинг 10.34. Разметка свойств `summary` и `description` в нашей рецензии

```
<p itemprop="summary">New York-style pizza right in historic downtown
Apex</p>
<p itemprop="description">
  Food is top-notch. Atmosphere is just right for a "neighborhood
  pizza joint." The restaurant itself is a bit cramped; if you're
  overweight, you may have difficulty getting in and out of your
  seat and navigating between other tables. Used to give free
  garlic knots when you sat down; now they give you plain bread
  and you have to pay for the good stuff. Overall, it's a winner.
</p>
```

Следите внимательно за изменениями в обоих файлах! Исходный вариант: **http://diveintohtml5.org/examples/review.html**, после добавления микроданных: **http://diveintohtml5.org/examples/review-plus-microdata.html**.

Свойства `location` и `geo` мы уже встречали, и они не отличаются от того, с чем мы имели дело раньше. Если вы еще только входите в курс дела, вернитесь к разделам, где обсуждались разметка адресов частных лиц и организаций, а также разметка информации географического позиционирования. Сама разметка представлена в листинге 10.35.

Листинг 10.35. Разметка свойств `location` и `geo`

```
<p itemprop="location" itemscope
        itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">100 North Salem Street</span><br>
  <span itemprop="locality">Apex</span>,
  <span itemprop="region">NC</span>
  <span itemprop="postal-code">27502</span><br>
  <span itemprop="country-name">USA</span>
</p>
<span itemprop="geo" itemscope
      itemtype="http://data-vocabulary.org/Geo">
  <meta itemprop="latitude" content="35.730796" />
  <meta itemprop="longitude" content="-78.851426" />
</span>
```

Следите внимательно за изменениями в обоих файлах! Исходный вариант: **http://diveintohtml5.org/examples/review.html**, после добавления микроданных: **http://diveintohtml5.org/examples/review-plus-microdata.html**.

Последняя строка представляет знакомую проблему: она содержит два информационных поля для одного элемента. Имя человека, написавшего рецензию — `Mark Pilgrim`, а написана эта рецензия 31 марта 2010 года. Как нам разметить эти два отличных друг от друга свойства? "Оберните" каждое свойство собственным элементом и поместите в каждое из них атрибут `itemprop`. Фактически, дата в этом примере должна быть размечена элементом `<time>` так, чтобы он представлял собой естественную "зацепку" для атрибута `itemprop`. Имя автора рецензии можно обернуть фиктивным элементом `<span>`. Вся разметка представлена в листинге 10.36.

Листинг 10.36. Разметка автора рецензии и времени ее написания

```
<p>- <span itemprop="reviewer">Mark Pilgrim</span>, last updated
   <time itemprop="dtreviewed" datetime="2010-03-31">
     March 31, 2010
   </time>
</p>
</article>
```

Следите внимательно за изменениями в обоих файлах! Исходный вариант: **http://diveintohtml5.org/examples/review.html**, после добавления микроданных: **http://diveintohtml5.org/examples/review-plus-microdata.html**.

Хорошо, а теперь давайте вернемся к обсуждению рейтингов. Самая сложная часть разметки рецензии — это рейтинг. По умолчанию рейтинги в словаре `Review` охватывают шкалу 1-5, где 1 соответствует оценке "ужасно", а 5 — "отлично". Если вы хотите использовать другую шкалу, вы, определенно, можете это сделать. Но сначала давайте обсудим стандартную шкалу.

```
<p>★★★★☆  (<span itemprop="rating">4</span> stars out of 5)</p>
```

Следите внимательно за изменениями в обоих файлах! Исходный вариант: **http://diveintohtml5.org/examples/review.html**, после добавления микроданных: **http://diveintohtml5.org/examples/review-plus-microdata.html**.

Если вы используете стандартную шкалу 1-5, то единственное свойство, которое вам нужно разметить — это сам рейтинг (который в рассматриваемом примере составляет 4. Но что, если вам хочется использовать другую шкалу? Вы можете так поступить, для этого вам нужно только декларировать пределы используемой шкалы. Например, выставлять оценки по шкале от 0 до 10, тогда вам по-прежнему нужно будет свойство `itemprop="rating"`, но вместо непосредственного выставления оценки вам потребуется использовать вложенный словарь **http://data-vocabulary.org/Rating**, чтобы объявить лучшую и худшую оценки по вашей индивидуальной шкале и выставить фактическую оценку в этих пределах. Вариант с разметкой по индивидуальной шкале рейтинга представлен в листинге 10.37.

Листинг 10.37. Вариант разметки рейтинга по индивидуальной шкале

```
<p itemprop="rating" itemscope
   itemtype="http://data-vocabulary.org/Rating">
  ★★★★★★★★★☆
  (<span itemprop="value">9</span> on a scale of
   <span itemprop="worst">0</span> to
   <span itemprop="best">10</span>)
</p>
```

Говоря простыми словами, эта разметка сообщает нам: "продукт, который я оцениваю, получает 9 баллов по шкале от 0 до 10".

Я уже упомянул, что микроданные рецензий влияют на результаты поиска? Да, они могут на это влиять. "Необработанные" данные, которые инструмент Google Rich Snippets извлечет из рецензии, снабженной микроданными, представлены в листинге 10.38.

Листинг 10.38. "Необработанные" данные, которые инструмент Google Rich Snippets извлечет из рецензии, снабженной микроданными

```
Item
  Type: http://data-vocabulary.org/Review
  itemreviewed = Anna's Pizzeria
```

```
  rating = 4
  summary = New York-style pizza right in historic downtown Apex
  description = Food is top-notch. Atmosphere is just right ...
  address = Item(__1)
  geo = Item(__2)
  reviewer = Mark Pilgrim
  dtreviewed = 2010-03-31

Item
  Id: __1
  Type: http://data-vocabulary.org/Organization
  street-address = 100 North Salem Street
  locality = Apex
  region = NC
  postal-code = 27502
  country-name = USA

Item
  Id: __2
  Type: http://data-vocabulary.org/Geo
  latitude = 35.730796
  longitude = -78.851426
```

Рис. 10.3. Пример того, как может выглядеть рецензия, когда она появится в результатах поиска

И вот как, с поправкой на пристрастия Google, фазы луны и так далее, может выглядеть рецензия, когда она появится в результатах поиска (рис. 10.3).

Рекомендации по дальнейшему чтению

Ресурсы, относящиеся к микроданным:

- Live microdata playground — **http://foolip.org/microdatajs/live/**
- HTML5 microdata specification — **http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#microdata**

Ресурсы, касающиеся Google Rich Snippets:

- About rich snippets and structured data — **https://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170**
- Marking up contact and social networking information — **https://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146646**
- Businesses & organizations — **https://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146861**
- Events — **https://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=164506**
- Reviews — **https://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146645**
- Review ratings — **https://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=172705**
- Google Rich Snippets Testing Tool — **http://www.google.com/webmasters/tools/richsnippets**
- Google Rich Snippets Tips and Tricks — **http://knol.google.com/k/google-rich-snippets-tips-and-tricks#**

Глава 11

Манипулирование историей — сочетаем приятное с полезным

Панель адреса браузера (location bar) — это, возможно, самая "хитроумная" часть пользовательского интерфейса из всех существующих. Иногда URL пишут на рекламных щитах, на общественном транспорте (автобусах, электричках), а иногда они попадаются даже в уличных граффити. В сочетании с кнопкой **Back** — пожалуй, самой важной во всем интерфейсе браузера — вы получаете мощное средство навигации по гигантскому морю взаимосвязанных ресурсов, называемому Всемирной Паутиной.

Интерфейс прикладного программирования истории HTML5 (**http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html**) предоставляет стандартизированный способ управления историей посещений страниц в браузере при помощи скриптов. Часть этого API — навигация по истории — была доступна и в более ранних версиях HTML. Новые части, введенные в HTML5, включают способы добавления записей в историю посещений браузера, позволяют видимым образом изменять URL в адресной строке браузера (без запуска обновления страницы) и предоставляют событие, которое срабатывает, когда эти записи удаляются пользователем из стека путем нажатия в браузере кнопки **Back**. Это значит, что URL в адресной строке браузера может продолжать выполнять свою работу как уникальный идентификатор ресурса, даже в приложениях, "под завязку" набитых скриптами, которые никогда не выполняют полного обновления страницы.

"Почему"

Зачем нужно вручную управлять адресом в браузере? В конце концов, переход по новому URL можно осуществлять с помощью простой ссылки; таким способом Web работает в течение уже более 20 лет. И Web продолжит работать таким способом. Данный API не делает попытки "подавить" Web. Как раз наоборот. В течение последних лет Web-разработчики нашли новые интересные способы управления Web без какой-либо помощи со стороны появляющихся стандартов. HTML5 history API разрабатывался с тем, чтобы гарантировать то, что URL останутся полезными даже в приложениях, перегруженных скриптами.

Возвращаясь к первоначальным принципам, давайте вспомним, а какую задачу выполняют URL? Здесь все ясно — URL идентифицирует уникальный ресурс. Вы можете дать на него непосредственную ссылку; вы можете поставить на него закладку; поисковые системы могут его проиндексировать; вы можете скопировать его, вставить в тело сообщения и отправить по электронной почте вашему адресату, который, получив письмо, сможет пройти по ссылке и просмотреть тот же ресурс, который изначально видели вы. Все это — отличные качества. Так что, URL очень важны.

Итак, нам нужны уникальные ресурсы, на которые указывают уникальные URL. Но, в то же самое время, браузеры всегда имели одно фундаментальное ограничение: если вы измените URL, даже через скрипт, то это запустит полное "обратное путешествие" на удаленный Web-сервер и полное обновление страницы. Это потребует времени и ресурсов и будет выглядеть особенно расточительным, когда вы переходите на страницу, которая в значительной мере аналогична текущей. Производится повторная загрузка всего, что присутствует на новой странице, даже тех ресурсов, которые не имеют никаких отличий от ресурсов, присутствующих на текущей странице. Нет никакой возможности сообщить браузеру о необходимости изменить URL, но загрузить, к примеру, только половину страницы.

Начиная с HTML5, history API позволяет вам это делать. Вместо того чтобы запускать процедуру полного обновления страницы, вы можете использовать скрипт, который сообщает браузеру о том, что необходимо загрузить только часть страницы. Понять эту концепцию нелегко, и это потребует некоторых усилий с вашей стороны. Так что, с этого момента читайте внимательно. Вы следите за ходом моей мысли?

Предположим, что у вас есть две страницы, страница A и страница B. Эти две страницы совпадают на 90%; только 10% содержимого одной страницы отличается от содержимого другой. Теперь представьте себе, что пользователь сначала посещает страницу A, после чего пытается перейти на страницу B. Но вместо запуска полного обновления всего содержимого страницы вы прерываете процесс перехода и вручную даете следующие указания:

1. *Поменять измененное содержимое* (с помощью `innerHTML` или других методов). *Загрузить 10% ресурсов со страницы B* (только то, что отличается от содержимого страницы A — возможно, с помощью `XMLHttpRequest`). Это потребует внесения некоторых изменений в ваше Web-приложение на серверной стороне. Вам потребуется написать код, который будет возвращать только те 10% ресурсов страницы B, которые отличаются от ресурсов страницы A. Это может быть скрытый URL или параметр запроса, который конечный пользователь обычно не видит.
2. *Поменять измененное содержимое* (с помощью `innerHTML` или других методов DOM). Возможно, вам также понадобятся обработчики событий для измененных элементов или другого заменяемого содержимого.
3. *Обновить адресную строку браузера, заменив ее на URL страницы B*, используя конкретный метод из HTML5 history API, о которых я вскоре расскажу.

В конце этой процедуры (если вы корректно ее выполните), браузер получит DOM, идентичную DOM страницы B, в точности такую же, как и та, что была бы получена при непосредственном переходе на страницу B. Адресная строка браузера будет содержать URL, идентичный URL страницы B, в точности такой же, как если бы вы непосредственно перешли на страницу B. Но при этом вы никогда не будете перемещаться непосредственно на страницу B, и вы никогда не будете выполнять полного обновления страницы. Все события, которые произойдут, будут иллюзорными. Но поскольку эта "скомпилированная" страница выглядит точно так же, как и "настоящая" страница B, и имеет URL, идентичный URL страницы B, пользователь никогда не заметит разницы (и не оценит вашей тяжелой работы по всему этому "микроменеджменту").

"Как"

HTML5 history API представляет собой просто набор методов, работающих с объектом `window.history`, плюс одно событие объекта `window`. Вы можете использовать их, чтобы выявить поддержку API истории HTML5 в вашем браузере (**http://diveintohtml5.org/detect.html#history**). На данный момент эта поддержка ограничивается лишь новейшими версиями в очень немногих браузерах, что относит обсуждаемые методы к лагерю "прогрессивного развития".

Краткая сводка данных о поддержке HTML5 history API в современных браузерах и мобильных устройствах приведена в табл. 11.1.

Таблица 11.1. *Поддержка* `history.pushState` ***в современных браузерах и мобильных устройствах***

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
·	4.0+	5.0+	8.0+	·	4.2.1+	·

Давайте рассмотрим демонстрационный пример Якоба Нильсена (Jakob Nielsen), который называется dive into dogs (**http://diveintohtml5.org/examples/history/fer.html**), и посмотрим, как это работает. dive into dogs — это простой, но нетривиальный пример использования HTML5 history API. Он демонстрирует общий подход: длинный элемент `<article>` с ассоциированной встроенной фотогалереей. В браузере, обеспечивающем поддержку API истории, щелчки по кнопкам **Next** и **Previous** в фотогалерее обновят текущую фотографию и, кроме того, обновят URL в адресной панели браузера, но не запустят полного обновления страницы. В браузерах, не поддерживающих history API — или в браузерах, обеспечивающих эту поддержку, но там, где пользователь блокировал поддержку скриптов — ссылки будут просто работать так, как обычно, в результате чего переход на новую страницу будет сопровождаться полным обновлением.

Соответствующая разметка для страницы с единственной фотографией приведена в листинге 11.1.

Листинг 11.1. Разметка страницы, содержащей фотогалерею с единственной фотографией

```
The pledge
<aside id="gallery">
  <p class="photonav">
    <a id="photonext" href="casey.html">Next &gt;</a>
    <a id="photoprev" href="adagio.html">&lt; Previous</a>
  </p>
  <figure id="photo">
    <img id="photoimg" src="gallery/1972-fer-500.jpg"
            alt="Fer" width="500" height="375">
    <figcaption>Fer, 1972</figcaption>
  </figure>
</aside>
```

Ничего необычного здесь нет. Сама фотография представляет собой элемент `<img>` внутри элемента `<figure>`, ссылки — это просто обычные элементы `<a>`, а весь код "обернут" элементом `<aside>`. Важно, что это совершенно обычные ссылки, работающие совершенно привычным образом, так, как обычно. Весь код, который следует за этой разметкой, представляет собой скрипт детекции (detection script). Если пользователь работает в браузере без поддержки history API, то никакой фрагмент кода, использующего history API, никогда не будет исполнен. И, конечно, всегда найдутся пользователи, у которых поддержка скриптов блокирована полностью.

Основная функция, `setupHistoryClicks()`, играющая роль "движущей силы", передает каждую из этих ссылок функции `addClicker()`, которая выполняет работу по установке индивидуального обработчика щелчка (листинг 11.2).

Листинг 11.2. Функция `setupHistoryClicks()`

```
function setupHistoryClicks() {
  addClicker(document.getElementById("photonext"));
  addClicker(document.getElementById("photoprev"));
}
```

Функция `addClicker()` принимает элемент `<a>` и добавляет обработчик события типа "щелчок мышью". И внутри этого обработчика ситуация как раз и становится интересной. Код функции `addClicker()` представлен в листинге 11.3.

Листинг 11.3. Код функции `addClicker()`

```
function addClicker(link) {
  link.addEventListener("click", function(e) {
    swapPhoto(link.href);
```

```
    history.pushState(null, null, link.href);
    e.preventDefault();
  }, false);
}
```

Функция `swapPhoto()` выполняет первые два шага нашего трехшагового "фокуса". Первая часть функции `swapPhoto()` берет часть URL навигационной ссылки — `casey.html`, `adagio.html`, и конструирует URL на скрытую страницу, которая не содержит более ничего, кроме разметки, необходимой для показа следующей фотографии. Код функции `swapPhoto()` приведен в листинге 11.4.

Листинг 11.4. Код функции `swapPhoto()`

```
function swapPhoto(href) {
  var req = new XMLHttpRequest();
  req.open("GET",
           "http://diveintohtml5.org/examples/history/gallery/" +
             href.split("/").pop(),
           false);
  req.send(null);
```

В листинге 11.5 приведен пример разметки, возвращенной **http://diveintohtml5.org/examples/history/gallery/casey.html**. Вы можете сверить ее с оригиналом, посетив URL непосредственно в браузере.

Листинг 11.5. Пример разметки, возвращенной http://diveintohtml5.org/examples/history/gallery/casey.html

```
<p class="photonav">
  <a id="photonext" href="brandy.html">Next &gt;</a>
  <a id="photoprev" href="fer.html">&lt; Previous</a>
</p>
<figure id="photo">
  <img id="photoimg" src="gallery/1984-casey-500.jpg"
         alt="Casey" width="500" height="375">
  <figcaption>Casey, 1984</figcaption>
</figure>
```

Выглядит ли эта разметка знакомой? Она должна! Это — та же самая базовая разметка оригинальной страницы, которая использовалась для отображения первой фотографии.

Вторая половина функции `swapPhoto()` выполняет второй шаг нашей трехшаговой комбинации: вставка этой вновь загруженной разметки на текущую страницу. Не забывайте, что здесь "обертка" `<aside>` заключает в себе все изображение, фотографию и надпись. Поэтому вставка разметки новой фотографии занимает одну

строку, устанавливая свойство `innerHTML` элемента `<aside>` на значение `responseText`, возвращенное из `XMLHttpRequest`, как показано в листинге 11.6.

Листинг 11.6. Вставка новой фотографии

```
  if (req.status == 200) {
    document.getElementById("gallery").innerHTML = req.responseText;
    setupHistoryClicks();
    return true;
  }
  return false;
}
```

Обратите также внимание на вызов `setupHistoryClicks()`. Это необходимо, чтобы сбросить индивидуальные обработчики щелчков мышью на вновь вставленные навигационные ссылки. Установка `innerHTML` стирает следы старых ссылок и их обработчиков событий.

Теперь вернемся к функции `addClicker()`. После успешной замены фотографий остается выполнить последний шаг нашей трехшаговой комбинации: установить URL в навигационной панели браузера без обновления страницы.

Наступает очередь функции `history.pushState(null, null, link.href)`.

Функция `history.pushState()` принимает три параметра:

1. Статус (`state`) может быть любой структурой данных JSON (JavaScript Object Notation)[1]. Он передается обратно обработчику события `popstate`, о котором я расскажу буквально через несколько минут. Нам не нужно отслеживать какой-либо статус в этом демонстрационном примере, поэтому я оставил для него значение `null`.
2. Параметр `title` может представлять собой любую строку. Большинство браузеров на текущий момент не используют этот параметр. Если вы хотите установить заголовок страницы, вам следует сохранить его в аргументе `state` и установить вручную в вашем обратном вызове `popstate`.
3. Аргумент `url` может представлять собой любой URL. Это — URL, который должен появляться в навигационной панели браузера.

Вызов `history.pushState` немедленно изменит URL в навигационной панели браузера. Итак, дошли ли мы до конца комбинации? Не совсем. Нам все еще нужно поговорить о том, что происходит, когда пользователь нажимает кнопку **Back**.

Как правило, когда пользователь переходит на новую страницу (с полным обновлением страницы), браузер заносит новый URL на стек истории, загружает новую страницу и визуализирует ее. Когда пользователь нажимает кнопку **Back**, браузер стаскивает со стека истории одну страницу и повторно отображает предыдущую страницу. Но что произойдет сейчас, когда вы "закоротили" эту навигацию,

[1] См. **http://en.wikipedia.org/wiki/JSON**, **http://www.hunlock.com/blogs/Mastering_JSON_(_JavaScript_Object_Notation_)**.

чтобы избежать полного обновления страницы? Хорошо, вы подделали "переход вперед" к новому URL. Теперь нам необходимо подменить и переход к предыдущему URL. Ключевую роль здесь играет событие `popstate`, как в листинге 11.7.

Листинг 11.7. "Подмена" перехода к предыдущему URL

```
window.addEventListener("popstate", function(e) {
    swapPhoto(location.pathname);
}
```

После того как вы воспользовались функцией `history.pushState()` для занесения "поддельного" URL на стек истории браузера, при нажатии пользователем кнопки **Back** браузер запустит событие `popstate` в отношении объекта `window`. Это — ваш шанс создать полную иллюзию однажды и навсегда. Потому что заставить что-то исчезнуть недостаточно, нужно позаботиться и о том, чтобы вернуть исчезнувшее "на место".

В данной демонстрации под словами "вернуть на место" подразумевается "заменить исходную фотографию", и мы это делаем, вызывая функцию `swapPhoto()` с текущим адресом. Ко времени обратного вызова `popstate`, URL, отображающийся в навигационной панели браузера, уже будет заменен предыдущим URL. Кроме того, глобальное свойство `location` уже будет обновлено предыдущим URL.

Чтобы вам понять, как все это происходит, я опишу весь "фокус" пошагово, с начала и до конца:

- Пользователь посещает страницу **http://diveintohtml5.org/examples/history/fer.html**, читает всю историю и видит фотографию собаки по имени Фер (Fer).
- Пользователь щелкает по ссылке **Next**, элемент `<a>`, свойство `href` которого выглядит как **http://diveintohtml5.org/examples/history/casey.html**.
- Вместо перехода на страницу **http://diveintohtml5.org/examples/history/casey.html** с полным обновлением всей страницы, индивидуальный обработчик щелчка мышью элемента `<a>` перехватывает щелчок и исполняет свой собственный код.
- Наш индивидуальный обработчик щелчка мышью вызывает функцию `swapPhoto()`, которая создает объект `XMLHttpRequest` для синхронной загрузки фрагмента HTML, расположенного по адресу **http://diveintohtml5.org/examples/history/gallery/casey.html**.
- Функция `swapPhoto()` устанавливает свойство `innerHTML` "обертки" фотогалереи (элемента `<aside>`), замещая таким образом подписанную фотографию Фер снабженным соответствующей подписью фото собаки Кэйси (Casey).
- Наконец, наш индивидуальный обработчик вызывает функцию `history.pushState()`, чтобы "вручную" заменить URL в навигационной панели браузера на **http://diveintohtml5.org/examples/history/casey.html**.
- Пользователь щелкает в браузере по кнопке **Back**.
- Браузер обнаруживает, что URL был "вручную" занесен на стек истории (функцией `history.pushState()`). Вместо перехода к прежнему URL с полным об-

новлением всей страницы, браузер просто обновляет навигационную панель, чтобы она отображала предыдущий URL (**http://diveintohtml5.org/examples/history/fer.html**), и запускает событие `popstate`.

- Наш индивидуальный обработчик события `popstate` снова вызывает функцию `swapPhoto()`, но в этот раз с предыдущим URL, который теперь отображается в навигационной панели браузера.
- Функция `swapPhoto()` с помощью `XMLHttpRequest` снова загружает фрагмент HTML, расположенный по адресу **http://diveintohtml5.org/examples/history/gallery/fer.html**, и устанавливает свойство `innerHTML` элемента `<aside>`, таким образом замещая фотографию Кэйси фотографией Фер. Обе фотографии снабжены соответствующими надписями.

Создается полная иллюзия переходов по ссылкам. Все видимые свидетельства осуществления перехода (содержимое страницы и URL в навигационной панели) говорят пользователям о том, что они посетили сначала одну страницу, потому другую, а потом опять вернулись на первую. Но при этом полного обновления страниц не происходит — все события, имеющие место, представляют собой искусно созданную иллюзию.

Материалы, рекомендуемые для дальнейшего чтения

- Раздел "Session history and navigation" в черновике стандарта HTML5 — **http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html**
- Статья "Manipulating the browser history" на сайте Центра разработки Mozilla — **https://developer.mozilla.org/en/DOM/Manipulating_the_browser_history**
- Демонстрация "Simple history API" — **http://html5demos.com/history**
- Демонстрация "20 Things I Learned About Browsers and the Web" — **http://www.20thingsilearned.com/**
- "Using HTML5 today" (**https://www.facebook.com/note.php?note_id=438532093919**) — описывает использование API-истории для Facebook (**https://www.facebook.com/**)
- "The Tree Slider" (**https://github.com/blog/760-the-tree-slider**) — описание использования истории API для Github (**https://github.com/**)
- History.js (**https://github.com/balupton/History.js/**), мета-API для манипулирования историей в новых и наследуемых браузерах

П Р И Л О Ж Е Н И Я

Приложение 1

Алфавитный справочник по выявлению поддержки всех функций HTML5

Вы все еще не слишком уверенно ориентируетесь в функциях HTML5? Затрудняетесь с выявлением их поддержки браузерами? Концептуальное введение в выявление поддержки функций HTML5 браузерами и мобильными устройствами можно найти *в главе 2* данной книги. Вам нужна единая всеобъемлющая библиотека? Попробуйте библиотеку `Modernizr` (**http://www.modernizr.com/**). В данном приложении будет приведен алфавитный список всех элементов.

Алфавитный список всех элементов

`<audio>`

(http://tinyurl.com/656y3my)

```
return !!document.createElement('audio').canPlayType;
```

`<audio>` в формате MP3

(https://secure.wikimedia.org/wikipedia/en/wiki/MP3)

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mpeg;').replace(/no/, ''));
```

`<audio>` в формате Vorbis

(https://secure.wikimedia.org/wikipedia/en/wiki/Vorbis)

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/ogg; co-
decs="vorbis"').replace(/no/, ''));
```

`<audio>` в формате WAV

(https://secure.wikimedia.org/wikipedia/en/wiki/WAV)

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/wav; co-
decs="1"').replace(/no/, ''));
```

`<audio>` в формате AAC

(**https://secure.wikimedia.org/wikipedia/en/wiki/Advanced_Audio_Coding**)

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mp4; co-
decs="mp4a.40.2"').replace(/no/, ''));
```

`<canvas>`

(**http://diveintohtml5.org/canvas.html**)

```
return !!document.createElement('canvas').getContext;
```

Текстовый API `<canvas>`

(**http://diveintohtml5.org/canvas.html**)

```
var c = document.createElement('canvas');
return c.getContext && typeof c.getContext('2d').fillText == 'function';
```

`<command>`

(**http://tinyurl.com/6aeqlcj**)

```
return 'type' in document.createElement('command');
```

`<datalist>`

(**http://tinyurl.com/68wvch2**)

```
return 'options' in document.createElement('datalist');
```

`<details>`

(**http://tinyurl.com/64ojbcx**)

```
return 'open' in document.createElement('details');
```

`<device>`

(**http://tinyurl.com/62ukn2p**)

```
return 'type' in document.createElement('device');
```

Валидизация ограничений `<form>`

(**http://tinyurl.com/652j85v**)

```
return 'noValidate' in document.createElement('form');
```

`<iframe sandbox>`

(**http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox**)

```
return 'sandbox' in document.createElement('iframe');
```

`<iframe srcdoc>`

(**http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox**)

```
return 'srcdoc' in document.createElement('iframe');
```

`<input autofocus>`

(**http://diveintohtml5.org/forms.html#autofocus**)

```
return 'autofocus' in document.createElement('input');
```

<input placeholder>

(http://diveintohtml5.org/forms.html#placeholder)

```
return 'placeholder' in document.createElement('input');
```

<textarea placeholder>

(http://diveintohtml5.org/forms.html#placeholder)

```
return 'placeholder' in document.createElement('textarea');
```

<input type="color">

(http://tinyurl.com/ycwmlxb)

```
var i = document.createElement('input');
i.setAttribute('type', 'color');
return i.type !== 'text';
```

<input type="email">

(http://diveintohtml5.org/forms.html#type-email)

```
var i = document.createElement('input');
i.setAttribute('type', 'email');
return i.type !== 'text';
```

<input type="number">

(http://diveintohtml5.org/forms.html#type-number)

```
var i = document.createElement('input');
i.setAttribute('type', 'number');
return i.type !== 'text';
```

<input type="range">

(http://diveintohtml5.org/forms.html#type-range)

```
var i = document.createElement('input');
i.setAttribute('type', 'range');
return i.type !== 'text';
```

<input type="search">

(http://diveintohtml5.org/forms.html#type-search)

```
var i = document.createElement('input');
i.setAttribute('type', 'search');
return i.type !== 'text';
```

<input type="tel">#

(http://tinyurl.com/5s83vyw)

```
var i = document.createElement('input');
i.setAttribute('type', 'tel');
return i.type !== 'text';
```

<input type="url">

(http://diveintohtml5.org/forms.html#type-url)

```
var i = document.createElement('input');
i.setAttribute('type', 'url');
return i.type !== 'text';
```

<input type="date">#

(http://diveintohtml5.org/forms.html#date)

```
var i = document.createElement('input');
i.setAttribute('type', 'date');
return i.type !== 'text';
```

<input type="time">

(http://diveintohtml5.org/forms.html#date)

```
var i = document.createElement('input');
i.setAttribute('type', 'time');
return i.type !== 'text';
```

<input type="datetime">

(http://diveintohtml5.org/forms.html#date)

```
var i = document.createElement('input');
i.setAttribute('type', 'datetime');
return i.type !== 'text';
```

<input type="datetime-local">

(http://diveintohtml5.org/forms.html#date)

```
var i = document.createElement('input');
i.setAttribute('type', 'datetime-local);
return i.type !== 'text';
```

<input type="month">

(http://diveintohtml5.org/forms.html#date)

```
var i = document.createElement('input');
i.setAttribute('type', 'month');
return i.type !== 'text';
```

<input type="week">

(http://diveintohtml5.org/forms.html#date)

```
var i = document.createElement('input');
i.setAttribute('type', 'week');
return i.type !== 'text';
```

<meter>

(http://tinyurl.com/6bn3hgw)

```
return 'value' in document.createElement('meter');
```

<output>

(http://tinyurl.com/6ew6zze)

```
return 'value' in document.createElement('output');
```

<progress>

(http://tinyurl.com/6be53bq)

```
return 'value' in document.createElement('progress');
```

<time>

(http://tinyurl.com/6z5hdju)

```
return 'valueAsDate' in document.createElement('time');
```

<video>

(http://diveintohtml5.org/video.html)

```
return !!document.createElement('video').canPlayType;
```

Титры <video>

(http://tinyurl.com/3yz6vqr)

```
return 'src' in document.createElement('track');
```

<video poster>

(http://tinyurl.com/5stjtxu)

```
return 'poster' in document.createElement('video');
```

<video> в формате WebM

(http://www.webmproject.org/)

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/webm; codecs="vp8, vor-
bis"').replace(/no/, ''));
```

<video> в формате H.264

(http://diveintohtml5.org/video.html#h264)

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/mp4; codecs="avc1.42E01E,
mp4a.40.2"').replace(/no/, ''));
```

<video> в формате format#

(http://diveintohtml5.org/video.html#theora)

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/ogg; co-
decs="theora"').replace(/no/, ''));
```

contentEditable

(http://tinyurl.com/68xx6rt)

```
return 'isContentEditable' in document.createElement('span');
```

Обмен сообщениями между документами

(http://tinyurl.com/5ualn9a)

```
return !!window.postMessage;
```

Drag-and-drop

(http://tinyurl.com/25eb3fe)

```
return 'draggable' in document.createElement('span');
```

File API

(http://dev.w3.org/2006/webapi/FileAPI/)

```
return typeof FileReader != 'undefined';
```

Geolocation

(http://diveintohtml5.org/geolocation.html)

```
return !!navigator.geolocation;
```

History

(http://tinyurl.com/63ux24v)

```
return !!(window.history && window.history.pushState);
```

IndexedDB

(http://www.w3.org/TR/IndexedDB/)

```
return !!window.indexedDB;
```

Local storage

(http://dev.w3.org/html5/webstorage/)

```
try {
  return 'localStorage' in window && window['localStorage'] !== null;
} catch(e) {
  return false;
}
```

Microdata

(http://tinyurl.com/6xe9t2g)

```
return !!document.getItems;
```

Offline web applications

(http://diveintohtml5.org/offline.html)

```
return !!window.applicationCache;
```

Server-sent events

(http://dev.w3.org/html5/eventsource/)

```
return typeof EventSource !== 'undefined';
```

Session storage

(http://dev.w3.org/html5/webstorage/)

```
try {
  return 'sessionStorage' in window && window['sessionStorage'] !== null;
} catch(e) {
  return false;
}
```

SVG

(http://www.w3.org/TR/SVG/)

```
return !!(document.createElementNS && docu-
ment.createElementNS('http://www.w3.org/2000/svg', 'svg').createSVGRect);
```

SVG в text/html

(https://hacks.mozilla.org/2010/05/firefox-4-the-html5-parser-inline-svg-speed-and-more/)

```
var e = document.createElement('div');
e.innerHTML = '<svg></svg>';
return !!(window.SVGSVGElement && e.firstChild instanceof win-
dow.SVGSVGElement);
```

Undo

(http://tinyurl.com/5u6l8nf)

```
return typeof UndoManager !== 'undefined';
```

Web Sockets

(http://dev.w3.org/html5/websockets/)

```
return !!window.WebSocket;
```

Web SQL Database

(http://dev.w3.org/html5/webdatabase/)

```
return !!window.openDatabase;
```

Web Workers

(http://www.whatwg.org/specs/web-workers/current-work/)

```
return !!window.Worker;
```

Widgets

(http://www.w3.org/TR/widgets/)

```
return typeof widget !== 'undefined';
```

XMLHttpRequest: кросс-доменные запросы

(http://tinyurl.com/6e423ey)

```
return "withCredentials" in new XMLHttpRequest;
```

XMLHttpRequest: отправить как данные формы

(**http://www.w3.org/TR/XMLHttpRequest2/#the-formdata-interface**)

```
return !!window.FormData;
```

XMLHttpRequest: upload progress events

(**http://www.w3.org/TR/XMLHttpRequest2/#the-upload-attribute**)

```
return "upload" in new XMLHttpRequest;
```

Материалы для дальнейшего чтения

Спецификации и стандарты:

- HTML5 — **http://www.whatwg.org/specs/web-apps/current-work/multipage/**
- Geolocation — **http://www.w3.org/TR/geolocation-API/**
- Server-Sent Events — **http://dev.w3.org/html5/eventsource/**
- WebSimpleDB — **http://dvcs.w3.org/hg/IndexedDB/raw-file/tip/Overview.html**
- Web Sockets — **http://dev.w3.org/html5/websockets/**
- Web SQL Database — **http://dev.w3.org/html5/webdatabase/**
- Web Storage — **http://dev.w3.org/html5/webstorage/**
- Web Workers — **http://www.whatwg.org/specs/web-workers/current-work/**
- Widgets — **http://www.w3.org/TR/widgets/**
- XMLHttpRequest Level 2 — **http://www.w3.org/TR/XMLHttpRequest2/**

Библиотеки JavaScript:

- Modernizr — **http://www.modernizr.com/**

Приложение 2

Краткая "шпаргалка" HTML5

Общие сокращения, встречающиеся в данном приложении:

`$new=document.createElement`

`$bool=function(any){return!(any=="no"||!any)}`

Поддержку большинства новых функций можно выявить через JavaScript. Чтобы протестировать поддержку видео в стандарте HTML5, создайте новый элемент `<video>` и проверьте в DOM следующее свойство:

`if("canPlayType" in $new("video")){...}` (см. *главу 2*: **http://diveintohtml5.org/detect.html**)

Новые элементы

См. *главу 3, "Что все это означает?"* (**http://diveintohtml5.org/semantics.html**)

Тестирование поддержки

`<command>` (**http://tinyurl.com/6aeqlcj**)	`"type"` в `$new("command")`
`<datalist>` (**http://tinyurl.com/68wvch2**)	`"options"` в `$new("datalist")`
`<details>` (**http://tinyurl.com/64ojbcx**)	`"open"` в `$new("details")`
`<output>` (**http://tinyurl.com/6ew6zze**)	`"value"` в `$new("output")`
`<progress>` (**http://tinyurl.com/create.php**)	`"value"` в `$new("progress")`
`<meter>` (**http://tinyurl.com/6bn3hgw**)	`"value"` в `$new("meter")`
`<time>` (**http://tinyurl.com/6z5hdju**)	`"valueAsDate"` в `$new("time")`

Текстовые аннотации: `<ruby>` (**http://tinyurl.com/69jqejf**), `<rt>` (**http://tinyurl.com/6afho5v**), `<rp>` (**http://tinyurl.com/6angb37**)

Семантика: элементы, которые могут использоваться во всех браузерах. IE 9 нуждается в промежуточном интерфейсе (https://code.google.com/p/html5shiv/)

`<article>` (**http://tinyurl.com/5wluqpy**), `<aside>` (**http://tinyurl.com/6c69kxn**), `<header>` (**http://tinyurl.com/6jxpybe**), `<hgroup>` (**http://tinyurl.com/6y2mkxp**), `<footer>` (**http://tinyurl.com/23yp7zh**), `<section>` (**http://tinyurl.com/467zfyl**),

`<nav>` (**http://tinyurl.com/28h5nwt**), `<figure>` (**http://tinyurl.com/6ar5vcp**), `<figcaption>` (**http://tinyurl.com/6kfka6v**), `<mark>` (**http://tinyurl.com/5wjy2a7**), `<summary>` (**http://tinyurl.com/6xq6cqk**)

Заново документированы: `<embed>` (**http://tinyurl.com/6adm63g**), `<keygen>` (**http://tinyurl.com/6djz5bo**), `<wbr>` (**http://tinyurl.com/2784hlp**)

Устаревшие элементы (**http://dev.w3.org/html5/html4-differences/#absent-elements**) — еще поддерживаются, но не валидизируются: `<basefont>`, `<big>`, `<center>`, `<font>`, `<s>`, `<strike>`, `<frame>`, `<frameset>`, `<noframes>`, `<applet>`, `<dir>`, `<isindex>`, `<tt>`, `<u>`, `<acronym>` (следует использовать элемент `<abbr>` (**http://tinyurl.com/67waabw**))

Всегда заключайте в кавычки свои значения атрибутов. Если вам так нравится, сохраняйте символы наклонной косой черты на конце. Валидатор по-прежнему крут: **http://html5.validator.nu/**.

Формы

Обеспечивается обратная совместимость с HTML 4. См. *главу 9* этой книги *"Форма безумия"* (**http://diveintohtml5.org/forms.html**).

Тестирование поддержки

Валидизация (**http://tinyurl.com/652j85v**)	`"noValidate"` в `$new("form")`
Ограничение по регулярным выражениям constraint (**http://tinyurl.com/652j85v**)	`"pattern"` в `$new("input")`
Замещающий текст (**http://diveintohtml5.org/forms.html#placeholder**)	`"placeholder"` в `$new("input")`
Автофокус (**http://diveintohtml5.org/forms.html#autofocus**)	`"autofocus"` в `$new("input")`
Обязательные для заполнения поля (**http://tinyurl.com/6l3n4b8**)	`"required"` в `$new("input")`

Новые типы ввода (**http://diveintohtml5.org/forms.html**) — браузеры могут индивидуально настраивать стили или методы ввода.

`type="search"`	поле поиска
`type="number"`	счетчик
`type="range"`	ползунковый регулятор
`type="color"`	элемент управления для выбора цвета
`type="tel"`	телефонный номер

`type="url"`	Web-адрес
`type="email"`	Адрес электронной почты
`type="date"/"time"/"month"/"week"/"datetime"`	Элементы управления для выбора даты и времени

Тестирование поддержки новых типов ввода (http://diveintohtml5.org/detect.html#input-types) — все типы выполняются по одному и тому же образцу:

```
function() {var i = $new("input");
  i.setAttribute("type", "search");
  return i.type !== "text";}
```

Некоторые браузеры декларируют поддержку определенных типов ввода, но не предлагают пользовательского интерфейса по умолчанию. Выявить это можно с помощью `Modernizr` (**http://www.modernizr.com/**).

Мультимедиа

Кодирование видео Theora выполняется с помощью Firefogg (**http://firefogg.org/**), H.264 — с помощью HandBrake (**http://handbrake.fr/**), различные форматы — с помощью Miro Video Encoder (**http://www.mirovideoconverter.com/**). См. *главу 5*, *"Видео в Web"* (**http://diveintohtml5.org/video.html**).

Поддержка элемента `<audio>` (**http://tinyurl.com/656y3my**)	`"canPlayType" в $new("audio")`
Vorbis (**http://en.wikipedia.org/wiki/Vorbis**)	`$bool($new("audio").canPlayType ('audio/ogg; codecs="vorbis"'))`
MP3 (**http://en.wikipedia.org/wiki/MP3**)	`$bool($new("audio").canPlayType ('audio/mpeg;'))`
AAC (**http://en.wikipedia.org/wiki/Advanced_Audio_Coding**)	`$bool($new("audio").canPlayType ('audio/mp4; codecs="mp4a.40.2"'))`
Поддержка элемента `<video>` (**http://tinyurl.com/2cg52ob**)	`canPlayType" в $new("video")`
WebM (**http://en.wikipedia.org/wiki/WebM**)	`$bool($new("video").canPlayType ('video/webm; codecs="vp8, vorbis"'))`
Theora (**http://en.wikipedia.org/wiki/Theora**)	`$bool($new("video").canPlayType ('video/ogg; codecs="theora"'))`
H.264 (**http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC**)	`$bool($new("video").canPlayType ('video/mp4; codecs="avc1.42E01E, mp4a.40.2"'))`

Свойства (**http://tinyurl.com/65s9pap**) — применимы и к элементам `<audio>`, и к элементам `<video>`.

`src` (**http://tinyurl.com/5umdody**)	`string`
`preload` (**http://tinyurl.com/6k9x7nb**)	`string`
`currentTime` (**http://tinyurl.com/5ufzk2z**)	в секундах
`initialTime` (**http://tinyurl.com/6xu6x3v**)	в секундах, read-only
`duration` (**http://tinyurl.com/62uho7m**)	в секундах, read-only
`startOffsetTime` (**http://tinyurl.com/6jrxzt9**)	`datetime`, read-only
`paused` (**http://tinyurl.com/62fnwwu**)	`boolean`
`ended` (**http://tinyurl.com/5sl26yd**)	`boolean`, read-only
`autoplay` (**http://tinyurl.com/6954zs6**)	`boolean`
`loop` (**http://tinyurl.com/6cdl7gy**)	`boolean`
`controls` (**http://tinyurl.com/676tc3d**)	`boolean`
`volume` (**http://tinyurl.com/65mlmgg**)	От .0 до 1.0, по умолчанию = 1.0
`muted` (**http://tinyurl.com/63c5pdc**	`boolean`
`playbackRate` (**http://tinyurl.com/5wjnv2s**)	по умолчанию = 1.0
`currentSrc` (**http://tinyurl.com/5rbovoz**)	`string`, read-only
`tracks` (**http://tinyurl.com/6x6vcvm**)	массив объектов `TimeTrack` (**http://tinyurl.com/5thbfuy**)
`buffered` (**http://tinyurl.com/6z489p6**)	объект TimeRanges (**http://tinyurl.com/6bkqmbc**), read-only
`played` (**http://tinyurl.com/62u5lhr**)	объект TimeRanges, read-only
`seekable` (**http://tinyurl.com/6ycz6mb**)	объект TimeRanges, read-only
`networkState` (**http://tinyurl.com/6eoah6v**)	нумерованное значение, read-only
`readyState` **http://tinyurl.com/6gyw6d2**)	нумерованное значение, read-only
`error.code` (**http://tinyurl.com/6zrg4en**)	нумерованное значение, read-only

В элементах `<audio>` or `<video>` перечисляйте несколько элементов `<source>` (**http://tinyurl.com/694jo77**). Браузеры с поддержкой HTML5 не визуализируют дочерних элементов элемента `<video>`, поэтому включите туда резервную поддержку Flash (**http://diveintohtml5.org/video.html#ie**). Аудио и видео должны обслуживаться с соответствующим типом MIME, поэтому проверьте ваши заголовки `Content-Type`!

Автономные приложения

См. *главу 8* этой книги *"Давайте возьмем все это в автономный режим"* (**http://diveintohtml5.org/offline.html**).

Тестирование поддержки (**http://diveintohtml5.org/detect.html#offline**)	`window.applicationCache`
`<html manifest>` (**http://tinyurl.com/679pc3e**)	Ссылка на манифест кэша
Разделы манифеста кэша (**http://tinyurl.com/679pc3e**)	
`CACHE:`	Всегда кэшируется. Символы шаблона не используются
`NETWORK:`	Никогда не кэшируется. Используются символы шаблона "*"
`FALLBACK:`	Пары; второй элемент пары используется в автономном режиме
События (**http://tinyurl.com/6c6tx2l**)	Первые четыре наиболее распространены
`checking`	Всегда идет первым
`downloading`	Найден манифест, извлечение ресурсов
`progress`	Извлечение ресурсов продолжается
`cached`	Все ресурсы кэшированы
`noupdate`	Манифест не изменился
`updateeady`	Вызов `swapCache()` для активации
`obsolete`	Манифест — сообщение 404 (или 410)
`error`	Где-то произошла ошибка, и все пошло не так

Семантика HTTP все еще применима к ресурсам, перечисленным в манифесте, поэтому проверьте ваши заголовки `Expires` и `Cache-Control`. Манифест должен обслуживаться с заголовком `text/cache-manifest`, поэтому проверьте и заголовки `Content-Type`. Если загрузка какого-нибудь из обязательных ресурсов завершится неудачей, приложение не сможет работать в автономном режиме.

Геопозиционирование

См. *главу 6* данной книги *"Вы находитесь здесь (как и все остальные)"* (**http://diveintohtml5.org/geolocation.html**). Предоставление в общий доступ информации о географическом местоположении пользователя всегда является делом добровольным.

Тестирование поддержки (**http://diveintohtml5.org/detect.html#geolocation**)	`navigator.geolocation`
Функции (**http://tinyurl.com/5rq3xql**)	
`Position getCurrentPosition(callback, err, opt);`	
`long watchPosition(callback, err, opt);`	
`void clearWatch(watchId);`	Вызывается при успехе
`void err(positionError);`	Вызывается при ошибке
Объект `PositionOptions` (**http://tinyurl.com/4ovaolq**)	
`timeout`	в миллисекундах
`maximumAge`	в миллисекундах
`enableHighAccuracy`	`true` или `false`
Объект `Position` **http://tinyurl.com/4ovaolq**)	(в обратном вызове) имеет свойства `timestamp` и `coords`
Объект `Coordinates` (**http://tinyurl.com/6klqu9s**)	Неподдерживаемые свойства будут установлены на `null`
`latitude`	в десятичных градусах
`longitude`	в десятичных градусах
`altitude`	высота в метрах над референц-эллипсоидом (**http://en.wikipedia.org/wiki/Reference_ellipsoid**)
`accuracy`	в метрах
`altitudeAccuracy`	в метрах
`heading`	в градусах по часовой стрелке, отсчитываемые от истинного направления северного меридиана
`speed`	в метрах в секунду
Объект `PositionError` (**http://tinyurl.com/62osl4n**) (в обратном вызове имеет сообщение и код: `TIMEOUT`, `POSITION_UNAVAILABLE`,`PERMISSION_DENIED` или `UNKNOWN_ERROR`	

Canvas

Траектории аналогичны рисованию карандашом, фактически на "холсте" (canvas) не отрисовывается ничего до тех пор, пока не будут вызваны методы `fill()` или `stroke()`! Подробнее см. *главу 4* данной книги *"Давайте назовем это "холстом" (поверхностью для рисования)"* (**http://diveintohtml5.org/canvas.html**).

Базовая поддержка (**http://diveintohtml5.org/detect.html#canvas**)	`"getContext" in $new("canvas")`
Текстовая поддержка (**http://diveintohtml5.org/detect.html#canvas-text**)	`typeof $new("canvas").fillText=="function"`
Функции	Корневой путь является неявным; подпути должны быть явными. `drawImage()` может отрисовывать видео или `<canvas>`
`beginPath();` (**http://tinyurl.com/6fy8554**)	`drawImage(image,dx,dy,dw,dh);` (**http://tinyurl.com/4qtrylx**)
`closePath();` (**http://tinyurl.com/6j4a4yx**)	`rotate(angle);` (**http://tinyurl.com/5u6g4kr**)
`moveTo(x,y);` (**http://tinyurl.com/5vdgeuv**)	`translate(x,y);` (**http://tinyurl.com/5varu9s**)
`lineTo(x,y);` (**http://tinyurl.com/66asb5a**)	`arcTo(x1,y1,x2,y2,radius);` (**http://tinyurl.com/4hqhovl**)
`rect(x,y,w,h);` (**http://tinyurl.com/4pq84re**)	`isPointInPath(x,y);` (**http://tinyurl.com/4koj2x7**)
`fill();` (**http://tinyurl.com/4fozmj6**)	`fillRect(x,y,w,h);` (**http://tinyurl.com/4nyvaof**)
`stroke();` (**http://tinyurl.com/4mxmm5y**)	`strokeRect(x,y,w,h);` (**http://tinyurl.com/5wrawfj**)
`clip();` (**http://tinyurl.com/6bheox4**)	`clearRect(x,y,w,h);` (**http://tinyurl.com/5utdy7n**)
`save();` (**http://tinyurl.com/66b4f2x**)	`setTransform(a,b,c,d,e,f);` (**http://tinyurl.com/6h67kb9**)
`restore();` (**http://tinyurl.com/5s7wygj**)	`transform(a,b,c,d,e,f);` (**http://tinyurl.com/6776mo4**)
`scale(x,y);` (**http://tinyurl.com/364j7sj**)	`createImageData(sw,sh);` (**http://tinyurl.com/6ev29qk**)
`quadraticCurveTo(cpx,cpy,x,y);` (**http://tinyurl.com/5rzgob7**)	`putImageDa-ta(imagedata,dx,dy,x,y,w,h);` (**http://tinyurl.com/4vg4gpc**)
`bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y);` (**http://tinyurl.com/63bdhh9**)	`fillText(text,x,y,maxWidth);` (**http://tinyurl.com/q9u8n4**)

`arc(x,y,radius,startAngle,endAngle,anticlockwise);` (**http://tinyurl.com/6677tc9**)	`strokeText(text,x,y,maxWidth);` (**http://tinyurl.com/6htnlrh**)
`getImageData(sx,sy,sw,sh);` (**http://tinyurl.com/4oo3abw**)	`measureText(text);` (**http://tinyurl.com/5w8yhlo**)
`createLinearGradient(x0,y0,x1,y1);` (**http://tinyurl.com/63hob4b**)	`createPattern(image,repetition);` (**http://tinyurl.com/6jl826e**)
`createRadialGradient(x0,y0,r0,x1,y1,r1);` (**http://tinyurl.com/6ft7rae**)	
Свойства	Все свойства перезаписываемы (read/write). `fillStyle` и `strokeStyle` can могут представлять собой градиенты или узоры
`fillStyle` (**http://tinyurl.com/4retvbo**)	Цвет CSS, `default = "black"`
`shadowColor` (**http://tinyurl.com/4lmcypn**)	Цвет CSS, `default = "black"`
`strokeStyle` (**http://tinyurl.com/4t8qpfy**)	Цвет CSS, `default = "black"`
`shadowOffsetX` (**http://tinyurl.com/635c2b5**)	В пикселах, `default = 0`
`font` (**http://tinyurl.com/5valp6a**)	Шрифт CSS, `default = "10px sans-serif"`
`shadowOffsetY` (**http://tinyurl.com/6j5wwyd**)	В пикселах, `default = 0`
`textAlign` (**http://tinyurl.com/5wxsmps**)	Перечисляемые, `default = "start"`
`shadowBlur` (**http://tinyurl.com/6dgpg8o**)	В пикселах, `default = 0`
`textBaseline` (**http://tinyurl.com/5sd2yut**)	Перечисляемые, `default = "alphabetic"`
`globalAlpha` (**http://tinyurl.com/5upaaq2**)	.0 (прозрачный) — 1.0 (непрозрачный)
`lineWidth` (**http://tinyurl.com/695our7**)	В пикселах, `default = 1`
`lineCap` (**http://tinyurl.com/6jj8ljg**)	Перечисляемые, `default = "butt"`
`lineJoin` (**http://tinyurl.com/5rtuzcj**)	Перечисляемые, `default = "miter"`
`miterLimit` (**http://tinyurl.com/6xy4hgx**)	`float`, `default = 10`

Изучите все о статусах! Статус элемента `<canvas>` (**http://tinyurl.com/6k7hot5**) включает траекторию обрезки (clipping path), все свойства и все трансформации. Функция `save()` заносит статус в стек, а функция `restore()` стаскивает его со стека.

Полезные мелочи

`Doctype` (**http://tinyurl.com/4cze5yq**)	`<!DOCTYPE html>` Запускает визуализацию на основе стандартов (**http://diveintohtml5.org/semantics.html#the-doctype**) во всех браузерах. В HTML5 отсутствует "quirks mode"
Кодировка текста (**http://tinyurl.com/5r74p62**)	`<meta charset="utf-8">` Всегда декларируйте кодировку (**http://tinyurl.com/5wugd48**), даже если считаете себя "звездой". UTF-8 — это всегда безопасный выбор
Необязательные закрывающие теги (**http://tinyurl.com/yfwzd9**).	`<html>`, `<head>`, `<body>`, `<li>`, `<p>`, `<dt>`, `<dd>`, `<colgroup>`, `<option>`, `<optgroup>`, `<rt>`, `<rp>`, `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`, `<td>`. Исключение: всегда закрывайте `<p>` перед `<table>`, чтобы не столкнуться со "странностями" IE
Необязательные открывающие теги ((**http://tinyurl.com/yfwzd9**).)	`<html>`, `<head>`, `<body>`, `<tbody>`, `<colgroup>`. Удивите своих друзей! Можно пропустить теги `<html>` и все же валидизироваться!
Новые атрибуты (**http://tinyurl.com/6kjtzks**)	`<a media>` (**http://tinyurl.com/66yessp**), `<a ping>` (**http://tinyurl.com/6fod8uc**), `<base target>` (**http://tinyurl.com/64nzeud**), `<style scoped>` (**http://tinyurl.com/6bqg5dv**), `<script async>` (**http://tinyurl.com/yzf3nvp**), `<ol reversed>` (**http://tinyurl.com/6dh448o**)
Различные тесты	См. **http://diveintohtml5.org/everything.html**
IndexedDB (**http://tinyurl.com/379qggk**)	`window.indexedDB`
contentEditable (**http://tinyurl.com/68xx6rt**)	`"isContentEditable" в $new("a")`
Web Workers (**http://tinyurl.com/62qgu2**)	`window.Worker`
Drag-and-drop (**http://tinyurl.com/25eb3fe**)	`"draggable" в $new("span")`
Web Sockets (**http://dev.w3.org/html5/websockets/**)	`window.WebSocket`
File API (**http://dev.w3.org/2006/webapi/FileAPI/**)	`typeof FileReader!=="undefined"`
X-doc messaging (**http://tinyurl.com/5ualn9a**)	`window.postMessage`
Undo history (**http://tinyurl.com/6emnuwd**)	`typeof UndoManager!=="undefined"`
Web SQL (**http://dev.w3.org/html5/webdatabase/**)	`window.openDatabase`

`<iframe sandbox>` (**http://tinyurl.com/y9ajj3k**)	`"sandbox" в $new("iframe")`
Web Storage (**http://dev.w3.org/html5/webstorage/**)	`"localStorage" в window && window["localStorage"] !== null` См. *главу 7* этой книги *"Прошлое, настоящее и будущее Web-приложений для хранения данных"* (**http://diveintohtml5.org/storage.html**)
History API (**http://tinyurl.com/63ux24v**)	`window.history && window.history.pushState` См. *главу 11* данной книги *"Манипулирование историей — сочетаем приятное с полезным"* (**http://diveintohtml5.org/history.html**)
Inline SVG (**http://tinyurl.com/2aspc2y**)	`function() { var e=$new("div"); e.innerHTML="<svg></svg>"; return window.SVGSVGElement && e.firstChild instanceof window.SVGSVGElement}`

Предметный указатель