HTML5
ДЛЯ ПРОФЕССИОНАЛОВ

Мощные инструменты для разработки современных веб-приложений

Питер Лабберс,
Брайан Олберс,
Фрэнк Салим

Москва · Санкт-Петербург · Киев
2011

ББК 32.973.26-018.2.75
Л12
УДК 681.3.07

Издательский дом "Вильямс"

Главный редактор *С.Н. Тригуб*

Зав. редакцией *В.Р. Гинзбург*

Перевод с английского и редакция канд. хим. наук *А.Г. Гузикевича*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Лабберс, Питер, **Олберс**, Брайан, **Салим**, Фрэнк.
Л12 HTML5 для профессионалов: мощные инструменты для разработки современных веб-приложений. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 272 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1715-7 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Питер Лабберс, Брайан Олберс, Фрэнк Салим

HTML5 для профессионалов: мощные инструменты для разработки современных веб-приложений

Литературный редактор *Е.П. Перестюк*
Верстка *О.В. Романенко*
Художественный редактор *Е.П. Дынник*
Корректор *Л.А. Гордиенко*

Подписано в печать 28.12.2010. Формат 70x100/16
Гарнитура Bookman. Печать офсетная
Усл. печ. л. 21,93. Уч.-изд. л. 16,5
Тираж 1500 экз. Заказ № 25056.

Отпечатано по технологии CtP
в ОАО "Печатный двор" им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1715-7 (рус.)
ISBN 978-1-4302-2790-8 (англ.)

Оглавление

Содержание

Предисловие

В июне 2004 года представители сообщества веб-разработчиков, крупнейшие производители браузеров и Консорциум W3C собрались в Сан-Хосе, шт. Калифорния, чтобы обсудить целесообразность пересмотра стандартов в связи с колоссальным всплеском интереса к веб-приложениям. К концу второго дня заседаний вопрос о том, следует ли расширить спецификации HTML и DOM для учета новых потребностей разработчиков, был поставлен на голосование. С точки зрения сегодняшнего дня результат анонимного голосования, зафиксированный в протоколах собрания, выглядит довольно забавным: "за — 8, против — 14".

Ввиду явного расхождения участников собрания во мнениях, двумя днями позже представители производителей браузеров основали новое сообщество WHATWG, возложившее на себя заботу о дальнейшем развитии HTML с учетом запросов разработчиков веб-приложений. Оставшийся на своих позициях Консорциум W3C продолжил работу по созданию спецификации XHTML2, но через пять лет все-таки прекратил деятельность в этом направлении и подключился к разработке спецификации HTML5 совместно с WHATWG.

С тех пор прошло шесть лет, и теперь все мы ощущаем на себе благотворное влияние тех идей, которые зародились в беспокойных умах специалистов, задумавших HTML5. С одной стороны, эта спецификация кодифицирует стандарты дефакто, которых многие придерживаются уже на протяжении нескольких лет, а с другой — создает основу для веб-приложений следующего поколения. Реализация рекомендаций спецификации на практике означает создание более привлекательного и быстро реагирующего интерфейса для пользователей Интернета и во многих случаях позволяет значительно сократить объем необходимого для этого программного кода.

В этой книге материал изложен таким образом, чтобы читатель смог как можно быстрее освоить концепции, предусматриваемые HTML5 и родственными спецификациями. Вы ознакомитесь со средствами поддержки необходимых инструментов, изучите типичные примеры их применения и разрешите для себя множество вопросов, ответы на которые вам не удастся найти в спецификациях. Приведенные в книге образцы кода не просто иллюстрируют тривиальные примеры использования отдельных инструментов HTML5, но позволяют пройти через все этапы создания полноценных веб-приложений. Я надеюсь, что эта книга окажется для вас полезной и что перспективы, которые открывают перед нами веб-приложения нового поколения, будут восторгать вас не меньше, чем меня.

Пол Айриш
Ведущий разработчик библиотеки Modernizr,
член групп по связи с сообществами пользователей Google Chrome и jQuery

Об авторах

Питер Лабберс — директор департамента документации и обучения в компании Kaazing. Питер — большой энтузиаст HTML5 и технологии WebSocket, часто выступает с докладами и ведет обучающие курсы по всему миру. До прихода в компанию Kaazing Питер в течение почти десяти лет работал архитектором информационных систем в компании Oracle, где занимался программными решениями, оформляемыми в виде патентных заявок, и написанием книг, многие из которых отмечены различными наградами. Будучи гражданином Голландии, служил в войсках специального назначения — частях "зеленых беретов" королевства Голландии. В настоящее время живет вблизи национального лесопарка в Тахо, шт. Калифорния, и в свободное от работы время участвует в супермарафонах. Связаться с ним можно на Twitter (`@peterlubbers`).

Брайан Олберс — вице-президент компании Kaazing. Его карьера в области веб-разработки, включая предыдущую должность заведующего отделом разработки в компании Oracle, охватывает 15-летний период. Брайан регулярно выступает с докладами на всевозможных конференциях, таких как Web 2.0 Expo, AJAX-World Expo и JavaOne, рассказывая в основном об интернет-технологиях и построении пользовательских интерфейсов. Уроженец Техаса, в настоящее время Брайан проживает в Калифорнии, стараясь при малейшей возможности улизнуть на Гавайи. Свободное время, если не удается расслабиться на пляже, Брайан часто проводит в одном из виртуальных миров.

Фрэнк Салим — один из инженеров, участвовавших в создании стратегии шлюза и клиента WebSocket в компании Kaazing. Фрэнк родился в Сан-Диего, а в настоящее время проживает в Маунтин-Вью, шт. Калифорния. Имеет ученую степень в области информатики, полученную в колледже г. Помона. Свободное от программирования время посвящает чтению, рисованию и катанию на роликах.

О техническом рецензенте

Пол Хейн (Paul Haine) — разработчик клиентского программного обеспечения, в настоящее время проживает в Лондоне и работает в газете Guardian. Адрес его персонального веб-сайта — `http://www.joeblade.com`.

Введение

Спецификация HTML5 еще совсем свежа. Точнее, работа над ней все еще продолжается. А по словам некоторых недоброжелательно настроенных по отношению к ней экспертов, прежде чем новый стандарт увидит свет, должно пройти еще не менее 10 лет!

Многие могут спросить: зачем в таком случае публиковать сейчас книгу, которая называется "HTML5 для профессионалов"? Ответ прост: затем, что для любого разработчика, мечтающего о создании таких веб-приложений, которые по многим параметрам превосходили бы все остальные, время HTML5 уже наступило. Авторы книги, которые сами используют технологии HTML5 и обучают этому других на протяжении двух с лишним лет, могут с уверенностью утверждать, что внедрение новых стандартов происходит с поразительной быстротой. Даже в процессе написания книги мы были вынуждены постоянно пересматривать наши таблицы данных о поддержке спецификаций различными браузерами и заново формулировать предположения относительно степени готовности тех или иных средств к использованию.

Большинство пользователей не в полной мере отдают себе отчет в том, какие возможности доступны им в используемых браузерах. Конечно, те незначительные улучшения пользовательского интерфейса, которые время от времени появляются в результате автоматического обновления браузера, не могут не бросаться в глаза. Но пользователь может даже не догадываться о том, что обновленная версия браузера предоставляет, например, возможность свободного рисования на холсте или обеспечивает передачу сообщений по сети в режиме реального времени.

Эта книга поможет вам открыть для себя весь спектр возможностей, предлагаемых HTML5.

Для кого предназначена эта книга

Эта книга ориентирована на опытных разработчиков веб-приложений, знакомых с программированием на JavaScript. Иными словами, элементарные сведения, относящиеся к процессу веб-разработки, здесь не приводятся. Существует множество ресурсов, с помощью которых вы сможете быстро изучить основы веб-программирования. Запомнив это, пойдем дальше. Если хотя бы один из приведенных ниже пунктов относится к вам, то эта книга поможет разобраться в сути интересующих вас вопросов и предоставит информацию, которую вы наверняка хотите найти.

- Иногда вы ловите себя на мысли: "Вот если бы мой браузер мог..."
- Иногда вам не терпится открыть в браузере исходный код страницы особо впечатлившего вас веб-сайта и исследовать его с помощью каких-либо инструментов разработчика.
- Вы с интересом читаете примечания к выпускам обновлений браузера, чтобы быть в курсе последних новинок.
- Вы ищете способы оптимизации или рационализации своих веб-приложений.
- Вы хотите настроить свой сайт так, чтобы обеспечить наиболее комфортные условия работы для пользователей, располагающих последними версиями браузеров.

Если любое из этих утверждений относится к вам, то эта книга будет хорошим подспорьем.

Там, где это необходимо, мы отмечаем те или иные ограничения поддержки, предоставляемой браузерами, однако наша цель состоит не в том, чтобы снабдить вас подробными инструкциями относительно того, как запустить приложение HTML5 на браузере десятилетней давности. Практика показала, что обходные пути решения проблем и базовая поддержка, предоставляемая браузерами, эволюционируют настолько быстрыми темпами, что книги наподобие этой — не самый лучший источник информации такого рода. Вместо этого мы сфокусировали внимание на спецификации HTML5 и способах ее применения. Подробные описания отдельных специфических приемов можно найти в Интернете, хотя со временем они утратят свою актуальность.

Структура книги

В одиннадцати главах книги описаны наиболее популярные и часто используемые интерфейсы прикладного программирования (API), применяемые в HTML5. В некоторых случаях примеры создавались на основе материала предыдущих глав, чтобы сделать иллюстрацию тех или иных возможностей более наглядной.

В главе 1 дается общий обзор старых и последних версий спецификации HTML5. Наряду с новыми высокоуровневыми семантическими дескрипторами здесь представлены базовые изменения и логические обоснования для всех последних нововведений в HTML5. С этим материалом стоит ознакомиться в первую очередь.

В главах 2 и 3 описаны новые визуальные и мультимедийные элементы. Основное внимание уделено поиску простейших способов повышения привлекательности внешнего вида пользовательского интерфейса, не требующих применения подключаемых модулей или взаимодействия с сервером.

Глава 4 познакомит вас с действительно новой возможностью, попытки эмуляции которой в прошлом наталкивались на серьезные трудности, — способностью приложения определять текущее местоположение пользователя для дополнительной настройки внешнего вида и поведения пользовательского интерфейса. В этой области защита личной информации приобретает особо важное значение, в связи с чем приводятся также рекомендации, позволяющие избежать возможных ловушек.

Следующие две главы посвящены описанию непрерывно развивающихся возможностей HTML5, обеспечивающих обмен сообщениями с другими сайтами и передачу потоковых данных приложению в реальном времени. Отличительной особенностью здесь является простота и минимальный объем служебного трафика. Обсуждаемые методики позволят упростить многие из чересчур сложных архитектур, развертываемых для этой цели в Интернете в настоящее время.

В главе 7 представлены как минимальные доработки, которые можно уже сегодня использовать в настольных или мобильных веб-приложениях, чтобы сделать их более удобными в работе, так и более серьезные изменения, предназначенные для обнаружения ошибок ввода данных на веб-странице в наиболее типичных сценариях использования.

В главах 8, 9 и 10 рассматривается внутренняя "начинка" приложений. Здесь описаны различные способы оптимизации существующей функциональности для повышения производительности приложений и эффективности управления данными.

Наконец, в главе 11 приведен краткий обзор перспективных новинок, появления которых следует ожидать в ближайшем будущем.

Коды примеров и веб-сайт книги

Исходный код примеров, рассмотренных в книге, доступен для загрузки в разделе Source Code на сайте издательства Apress:

`http://www.apress.com/book/view/1430227907`

Кроме того, для книги создан сопутствующий сайт `www.prohtml5.com`, посетив который вы также сможете загрузить исходный код примеров.

Связь с авторами

Благодарим вас за покупку этой книги. Надеемся, что читать ее вам будет приятно и вы найдете в ней много полезного материала. Несмотря на все наши старания, в книге могли остаться незамеченными разного рода ошибки и опечатки, за которые мы заранее приносим свои извинения. Мы ждем ваших отзывов, вопросов и комментариев относительно содержания книги и исходного кода. Чтобы связаться с нами, воспользуйтесь следующим адресом электронной почты:

`prohtml5@gmail.com`

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: `info@williamspublishing.com`
WWW: `http://www.williamspublishing.com`

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1
в Украине: 03150, Киев, а/я 152

Глава 1

Обзор HTML5

Эта книга посвящена программированию с использованием средств HTML5, и поэтому вам прежде всего необходимо понять, что собой представляет язык HTML5, что ему предшествовало и какие новые по сравнению с HTML 4 возможности он предлагает.

В этой главе мы сразу же приступим к рассмотрению тех практических вопросов, ответы на которые интересуют всех. Чем замечателен язык HTML5 и почему в последнее время он привлекает к себе всеобщее внимание? Какие новые принципы проектирования делают его поистине революционным и вместе с тем необычайно легким для восприятия? Что стоит за парадигмой, отвергающей использование подключаемых модулей; что наследует HTML5 от прошлых версий? Что нового сейчас в HTML и способно ли его дальнейшее усовершенствование дать толчок началу новой эры в области разработки веб-приложений? Итак, приступим!

История появления HTML5

Стандарт HTML прошел долгий эволюционный путь. Его первая версия была опубликована в 1993 году в виде черновой спецификации для Интернета. В 90-е годы HTML оказался в центре всеобщего внимания. Очередные версии быстро сменяли друг друга. Вскоре после выпуска версии 2.0 вышла версия 3.2, затем — 4.0 (и это на протяжении всего лишь одного года!) и, наконец, в 1999 году — версия HTML 4.01. Контроль за разработкой соответствующих спецификаций по мере развития языка осуществляет международный консорциум W3C (World Wide Web Consortium).

Период стремительного развития сменился затишьем, и будущее HTML стало казаться довольно туманным. Фокус веб-стандартизации сместился в сторону XML и XHTML, в то время как HTML оказался оттесненным на задний план. Однако HTML вовсе не собирался умирать, и на нем по-прежнему базировалась значительная часть веб-контента. Для перехода к веб-приложениям нового типа и преодоления имеющихся недостатков HTML нуждался в новых средствах и спецификациях.

В 2004 году по инициативе небольшой группы специалистов, представлявших ряд крупных производителей браузеров и заинтересованных в поднятии веб-платформы на новый технологический уровень, было основано сообщество WHATWG (Web Hypertext Application Working Group). Благодаря усилиям его участников и появилась спецификация HTML5. Они же инициировали и разработку новых средств, специально предназначенных для веб-приложений, поскольку ясно осознавали, насколько велика потребность в таких средствах. Именно в то время все чаще стали употреблять термин "Web 2.0". Все это давало основания полагать, что мы действительно являемся свидетелями *второго* возрождения Интернета, если первым считать период, когда статические веб-сайты уступили место более динамичным и социально направленным веб-сайтам, нуждавшимся в гораздо большем разнообразии различных средств в своем арсенале.

В 2006 году к работе над HTML вновь подключился Консорциум W3C, в результате чего в 2008 году увидел свет первый рабочий вариант проекта рекомендаций по HTML 5[1], тогда как рабочая группа, занимавшаяся до этого разработкой спецификации XHTML 2, в 2009 году прекратила свою деятельность. Прошел еще один год, и вот наступил сегодняшний день. Поскольку HTML5 (как вскоре будет показано) решает ряд важных практических проблем, производители браузеров ведут активную работу по внедрению новых возможностей, и это несмотря на то, что до выпуска окончательного варианта спецификации еще очень далеко. В свою очередь, апробация спецификации в эксплуатируемых браузерах служит источником ценной информации, которая используется для улучшения самой спецификации. HTML5 быстро развивается, шагая в ногу со все возрастающими требованиями к веб-платформам.

Мысли вслух об HTML

Говорит Брайан: "Привет, меня зовут Брайан, и я с полным правом могу называть себя ветераном HTML.

Свою первую домашнюю страницу я создал в далеком 1995 году. В те времена домашние страницы использовались их авторами главным образом для того, чтобы рассказать о себе. Типичная страница состояла из нескольких неважно отсканированных изображений, тегов `<blink>` и краткой информации о том, где живет автор, что читает и над каким компьютерным проектом сейчас работает. Большинство моих приятелей "веб-разработчиков" либо учились, либо работали в университетах.

HTML тогда был довольно примитивен, а инструментарий для работы с ним отсутствовал. Веб-приложения, включающие в себя нечто большее, чем тривиальные сценарии для обработки текста, были редкостью. Код страниц создавался вручную с использованием любого удобного текстового редактора. Страницы обновлялись не чаще одного раза в несколько недель или даже месяцев, если это вообще делалось.

С тех пор прошло целых пятнадцать лет.

Сегодня многократное обновление пользователями своего сетевого профиля в течение дня совсем не редкость. Подобный тип интерактивного взаимодействия не мог бы стать реальностью, если бы не постоянное и неуклонное совершенствование онлайновых инструментальных средств, каждое последующее поколение которых лучше предыдущего.

Не забывайте об этом, когда будете читать книгу. Возможно, некоторые из приведенных в ней примеров покажутся вам слишком простыми, но открывающиеся за ними горизонты поистине безграничны. Те из нас, кто впервые использовал теги `<img>` в середине 1990-х годов, пожалуй, и не мечтали, что лет через десять станет возможным хранить и редактировать фотографии непосредственно в Интернете, хотя это и можно было предвидеть.

Авторы надеются, что рассмотренные в книге примеры вдохновят вас на дальнейшую деятельность в этой области и пробудят желание принять участие в закладке фундамента для "нового" Интернета следующего десятилетия."

Миф о 2022 годе и как к нему относиться

По состоянию на сегодняшний день спецификация HTML5 существует в виде *рабочего проекта* (Working Draft) и еще далека до завершения. Так когда же можно ожидать появления ее окончательного варианта? Вот некоторые ключевые даты, о которых вам будет нелишне знать. Первая из них — это 2012 год, на который за-

---

[1] Обратите внимание на наличие пробела перед номером версии. В названии спецификации, разрабатываемой WHATWG, этот пробел отсутствует, и использовать следует именно данный вариант написания. Подробный и увлекательный рассказ обо всех перипетиях развития HTML5 и характере взаимного сотрудничества W3C и WHATWG можно найти по адресу `http://www.habrahabr.ru/blogs/webstandards/103256/`. — *Примеч. ред.*

планирован выпуск *возможной рекомендации* (Candidate Recommendation). Вторая опорная точка — 2022 год, когда должна выйти *предлагаемая рекомендация* (Proposed Recommendation). Постойте-ка, не торопитесь! Не спешите захлопывать книгу, откладывая ее в сторону на 10 лет, чтобы она дожидалась своего часа, пока до конца не разберетесь, что на самом деле означают эти даты.

Пожалуй, более важной для нас является первая из указанных дат, она же и ближайшая, поскольку к этому времени спецификация HTML5 должна быть окончательно сформулирована, и тогда можно будет считать, что стандарт как таковой готов. Это произойдет примерно через два года. Важность же утверждения спецификации в статусе предлагаемой рекомендации (а этот момент наступит, с чем все мы согласимся, довольно не скоро) заключается в том, что к тому времени браузеры, как предполагается, будут предоставлять полную поддержку всех возможностей языка — благородная цель, достижение которой к 2022 году представляется довольно амбициозной задачей. Ведь если говорить честно, то в случае спецификации HTML 4 мы к этому даже не приблизились.

Действительно важным для нас является тот факт, что в настоящее время производители браузеров активно внедряют поддержку многих новых возможностей. В зависимости от того, какова ваша аудитория, вы можете использовать эти возможности уже сегодня. Несомненно, со временем придется то и дело вносить некоторые незначительные поправки, но это совсем небольшая плата за те преимущества, которые сулит доступ к передовым технологиям. Конечно, если большая часть вашей аудитории все еще пользуется браузером Internet Explorer 6.0, то многие из новейших средств не будут работать, и их придется эмулировать, однако это не является веским аргументом в пользу того, чтобы отказываться от HTML5. Ведь рано или поздно такие пользователи все равно перейдут к более новым версиям программ. Вполне возможно, что многие из них уже сейчас предпочтут установить у себя Internet Explorer 9.0, тем более что корпорация Microsoft обещает усиленную поддержку HTML5 в этой версии своего браузера. С практической точки зрения новые браузеры в сочетании с усовершенствованными технологиями эмуляции позволяют использовать многие средства HTML5 уже сегодня или, по крайней мере, в ближайшем будущем.

Кто занимается разработкой HTML5

Все мы хорошо знаем, что любое крупное начинание нуждается в определенной координации действий, и поэтому совершенно очевидно, что разработку спецификации HTML5 должен был кто-то возглавить. Эти обязанности возложили на себя три важных органа.

- *Web Hypertext Application Technology Working Group (WHATWG*[2]*)* (Рабочая группа по развитию технологии гипертекстовых веб-приложений). Основанная в 2004 году по инициативе представителей таких компаний-производителей браузеров, как Apple, Mozilla, Google и Opera, WHATWG разрабатывает спецификации HTML и программные интерфейсы для веб-приложений и способствует развитию открытого сотрудничества между производителями браузеров и другими заинтересованными сторонами.

---

[2] Любопытно отметить, что, в отличие от Консорциума W3C, решения которого принимаются голосованием, в WHATWG принят другой подход, не основанный на достижении консенсуса, и последнее слово всегда остается за главным редактором — Яном Хиксоном (Ian Hickson). — *Примеч. ред.*

- *World Wide Web Consortium (W3C)* (Консорциум Всемирной паутины, Консорциум W3C). В состав Консорциума W3C входит рабочая группа HTML, которая в настоящее время отвечает за разработку спецификации HTML5.
- *Internet Engineering Task Force (IETF)* (Специальная комиссия интернет-разработок). Комиссия состоит из рабочих групп, занимающихся такими протоколами Интернета, как HTML. HTML5 определяет новый программный интерфейс WebSocket, основанный на новом протоколе WebSocket, разработка которого поручена одной из рабочих групп IETF.

Новая идеология

Разработка HTML5 базируется на ряде принципов проектирования, сформулированных в отдельной спецификации WHATWG[3], которые в полном смысле слова олицетворяют собой новое отношение к практической реализации предлагаемых возможностей:

- совместимость;
- удобство в использовании;
- независимость от типа браузера;
- универсальность доступа.

Совместимость: идем проторенным путем

Не беспокойтесь — HTML5 не внесет неразберихи, свойственной любой революции. Скорее, наоборот — обеспечение безотказной работы приложений является одним из ключевых принципов этой спецификации. Если окажется, что какие-то средства HTML5 не поддерживаются, то сбоя не произойдет, потому что система сможет продолжить работу в режиме *корректного сокращения возможностей* (graceful degradation), используя только те средства языка, которые ей фактически доступны. Кроме того, поскольку информация в виде HTML-содержимого накапливалась в течение примерно 20 лет, то важное значение приобретает поддержка уже существующего контента.

На выявление и изучение характерных общих тенденций в имеющемся HTML-содержимом затрачиваются огромные усилия. Так, компания Google, исследовавшая частотную повторяемость наиболее употребительных имен идентификаторов в дескрипторах `DIV`, проанализировала миллионы веб-страниц и обнаружила, что некоторые из имен встречаются намного чаще других. Например, для обозначения заголовка содержимого часто используется элемент `DIV id="header"`. Мы уже подчеркивали, что HTML5 весьма прагматичен и ориентирован на решение практических проблем, не правда ли? Так не проще ли сразу предусмотреть в языке элемент `<header>`?

Несмотря на то что многие новшества HTML носят довольно революционный характер, все же более подходящим для него определением будет не "революция", а "эволюция". В конце концов, имеет ли смысл заново изобретать велосипед? (Правда, если такое желание у вас есть, то хотя бы предложите намного лучший его вариант!) Если какой-либо способ решения определенных задач ранее уже получил

---

[3] `http://www.w3.org/TR/html-design-principles`. Аналогичные принципы лежат в основе создания так называемых *микроформатов* — открытых стандартов форматов данных, которые, в частности, активно применяются для оптимального структурирования блогов (`http://www.microformats.org/about`). — *Примеч. ред.*

широкое распространение, то почему бы не пойти проторенным путем и не включить его в спецификацию?

Удобство в использовании и принцип приоритетности

Процесс разработки спецификации HTML5 основывается на принципе очередности приоритетов всех заинтересованных сторон, и эта очередность такова, что наивысший приоритет отдается пользователям. Это означает, что при наличии любых сомнений принимаемые решения должны в первую очередь учитывать интересы пользователей, затем — разработчиков, после этого — реализаторов (производителей браузеров) и, наконец, органов, ответственных за разработку спецификации (W3C/WHATWG). Соображениям же "теоретической чистоты" в этом параде приоритетов отводится самое последнее место. Как следствие, спецификация HTML5 имеет преимущественно практическую направленность, хотя кое в чем она еще далеко не совершенна.

Рассмотрим конкретный пример. В HTML5 приведенные ниже фрагменты кода равносильны.

```
id="prohtml5"
id=prohtml5
ID="prohtml5"
```

Безусловно, подобная вольность синтаксиса может вызывать возражения, но конечного пользователя все эти тонкости по большому счету не интересуют. Никто не собирается поощрять небрежность при написании кода, но если в данном случае потребовать соблюдения только какой-то одной из приведенных выше форм написания, то подобная строгость в некоторой степени сыграет против пользователя.

Разработка спецификации HTML5 также инициировала создание спецификации XHTML5, используя которую комплекты инструментальных средств для работы с XML смогут генерировать правильный HTML-код. Сериализация HTML- и XHTML-версий документа должна давать аналогичные DOM-деревья с минимальными различиями между ними. Очевидно, что синтаксис XHTML намного более строг, и код приведенных выше последних двух строк примера в нем был бы недопустимым.

Встроенная модель безопасности

Большое внимание уделяется тому, чтобы спецификация HTML5 обеспечивала безопасность уже сама по себе, без привлечения средств сторонних разработчиков. В каждой части спецификации содержится раздел, посвященный вопросам безопасности, причем эти вопросы всегда находятся на первом плане. HTML5 вводит новую модель безопасности, основанную на понятии *источника* (origin), которая не только проста в применении, но и последовательно используется в различных библиотеках функций. С помощью этой модели безопасности удается делать то, что раньше, как правило, оказывалось невозможным. Она позволяет организовать безопасную передачу данных между доменами, не прибегая к использованию всякого рода нестандартных, остроумных, но крайне ненадежных в плане безопасности решений. В этом отношении мы уж точно не будем с грустью вздыхать, вспоминая о старых добрых временах.

Разделение представления и содержимого

Спецификация HTML5 знаменует гигантский шаг вперед в отношении последовательного разделения представления и содержимого. В HTML5 это разделение обеспечивается везде, где только возможно, причем это достигается за счет использования CSS. В действительности значительная доля средств представления содержимого, предусмотренных в более ранних версиях HTML, больше не поддер-

живается, но благодаря соблюдению уже упоминавшегося ранее принципа совместимости эти средства по-прежнему будут работать. Вместе с тем, сама по себе эта идея далеко не нова и была взята на вооружение уже в процессе работы над спецификациями HTML4 Transitional и XHTML 1.1. В течение длительного времени она используется веб-дизайнерами в качестве одного из эталонов хорошей практики программирования, но в настоящее время значимость неукоснительного отделения представления от содержимого еще более возросла. С разметкой, обеспечивающей представление содержимого, связаны следующие проблемы:

- недостаточная доступность содержимого для людей с ограниченными возможностями;
- излишняя сложность (включение строковых стилей значительно затрудняет чтение кода);
- увеличение размера документа (в связи с повторением стилевого содержимого), что замедляет загрузку страниц.

Упрощение взаимодействия с браузерами

Все в HTML5 нацелено на максимальное упрощение и избежание ненужной сложности. Хотите знать основной девиз HTML5? "Проще — лучше! Упрощать везде, где только возможно". Вот лишь некоторые примеры реализации этих принципов:

- встроенные возможности браузера взамен сложного кода JavaScript;
- новый упрощенный дескриптор `DOCTYPE`;
- новое упрощенное объявление набора символов;
- мощные, но вместе с тем простые программные интерфейсы HTML5.

Кое-что из вышеперечисленного будет обсуждаться нами более подробно.

Отмеченная простота далась за счет увеличения размера спецификации, поскольку для этого потребовалась гораздо большая детализация формулировок, чем в любой другой предыдущей спецификации HTML. Чтобы успеть обеспечить к 2022 году истинную независимость от типа браузера, в спецификацию HTML5 неизбежно требовалось включить бесчисленные описания вариантов допустимого поведения браузеров. Любая неопределенность сделала бы достижение поставленной цели просто невозможным.

Другой причиной большей детализации спецификации HTML5 по сравнению с ее предшественницами является стремление предотвратить возможность неправильного истолкования ее положений. Определения отличаются предельной четкостью формулировок, особенно во всем, что касается веб-приложений. Поэтому не удивительно, что объем спецификации составляет свыше 900 страниц!

Кроме того, спецификация HTML5 предусматривает обеспечение безотказной работы приложений в соответствии с целым рядом усовершенствованных и весьма неординарных стратегий обработки ошибок. Вместо полного прекращения работы приложения в случае некорректного кода используется более прагматичный процесс *корректной обработки ошибок* (graceful error recovery), опять-таки с учетом того, что интересы конечного пользователя имеют наивысший приоритет. Например, ошибки в документах не будут приводить к катастрофическим сбоям, препятствующим отображению страниц. Для подобных случаев определены точные процедуры восстановления после сбоев, благодаря чему, в частности, браузеры смогут отображать "некорректную" разметку неким стандартным образом.

Универсальность доступа

Этот принцип базируется на трех подчиненных понятиях.

- *Доступность.* Чтобы обеспечить поддержку пользователей с ограниченными возможностями, HTML5 тесно взаимодействует с родственным стандартом — Web Accessibility Initiative (WAI) Accessible Rich Internet Applications (ARIA). Теперь в HTML-элементы можно добавлять роли WAI-ARIA, которые поддерживаются экранными считывателями.
- *Независимость от мультимедийных устройств.* Функциональность HTML5 должна обеспечиваться на любых устройствах и платформах, насколько это вообще возможно.
- *Поддержка всех международных языков.* Например, новый элемент `<ruby>` поддерживает аннотацию Ruby, применяемую для передачи на письме произношения слов в языках ряда стран Восточной Азии.

Парадигма без подключаемых модулей

HTML5 самостоятельно поддерживает многие возможности (например, встроенную библиотеку функций рисования, встроенные сокеты и т.д.), доступ к которым ранее требовал использования дополнительно подключаемых модулей (плагинов) или применения нестандартных способов модификации программ. С использованием подключаемых модулей связан целый ряд проблем, а именно:

- подключаемые модули не всегда могут быть установлены;
- использование подключаемых модулей может быть не разрешено или заблокировано (например, подключаемый модуль Flash не входит в состав стандартной поставки Apple iPad);
- подключаемые модули создают дополнительные риски для атак;
- подключаемые модули плохо интегрируются с остальной частью страницы (проблемы границ модуля, отсечения и прозрачности).

Несмотря на то что некоторые модули в большинстве случаев устанавливаются пользователями, в контролируемой корпоративной среде их часто блокируют. Кроме того, часть пользователей предпочитает запрещать использование дополнительных модулей из-за сопутствующей нежелательной рекламы. Но если пользователь отключит ваш модуль, то тем самым он сделает невозможным и использование программы, ответственной за отображение содержимого, которое вы хотите донести до пользователя.

Подключаемые модули часто сталкиваются с трудностями при попытках встроить свое содержимое в другой блок, отображаемый в браузере, что может приводить к проблемам с отсечением и прозрачностью на некоторых сайтах. Поскольку подключаемые модели используют собственную модель визуализации, отличную от той, которая используется на базовой веб-странице, разработчики могут испытывать затруднения в тех случаях, когда всплывающие меню или другие необходимые визуальные элементы пересекают границы модуля на странице. И тут наступает звездный час HTML5, который с помощью *собственной* функциональности решает все проблемы. Для стилевого оформления элементов можно использовать CSS и JavaScript. Фактически именно при решении этих задач HTML5 начинает поигрывать самыми большими своими мускулами, демонстрируя силу, которой предыдущие версии HTML не обладали. Речь идет не только о новой функциональности, предоставляемой новыми элементами, а о добавлении собственных воз-

можностей взаимодействия со сценариями и каскадными таблицами стилей, что позволяет решать такие задачи, которые ранее были просто недоступны.

Возьмем, к примеру, элемент `canvas` (холст, полотно). С его помощью вы можете выполнять операции, которые ранее были трудно осуществимы (попробуйте, например, провести диагональную линию на веб-странице средствами HTML 4). Гораздо более интересны возможности, предоставляемые библиотеками и средствами стилевого оформления, для использования которых достаточно нескольких строк CSS-кода. Помимо всего прочего, элементы HTML5, подобно послушным детям, хорошо уживаются друг с другом. Например, если захватить кадр из элемента `video` и отобразить его на холсте, то для того, чтобы воспроизвести видео, пользователю потребуется всего лишь выполнить на нем щелчок. Это лишь один из возможных примеров, демонстрирующих преимущества собственного кода по сравнению с подключаемыми модулями. В действительности, когда вы избавляетесь от необходимости работать с "черным ящиком", для вас упрощается практически *все*. В итоге вы получаете действительно мощную новую среду, и именно поэтому мы решили написать книгу, которая не только рассказывает о новых элементах, но и учит *программированию* с использованием средств HTML5.

Что включено в HTML5, а что не включено

Что же на самом деле включает в себя HTML5? Даже при внимательном чтении спецификации вы не обнаружите в ней упоминаний о некоторых из средств, описанных в данной книге. Например, в ней отсутствуют разделы HTML5 Geolocation и Web Workers. Получается, будто авторы обманывают вас или вводят в заблуждение? Вовсе нет! Первоначально многие разделы проекта HTML5 (например, Web Storage и Canvas 2D) были отдельными разделами спецификации HTML5, но затем, стремясь обеспечить цельность спецификации, их отнесли в документам других стандартов. Было решено, что целесообразно сначала обсудить и отредактировать положения, касающиеся таких средств, по отдельности и только потом включить их в официальные спецификации. Это позволяет избежать ситуаций, в которых наличие одного незначительного спорного элемента разметки тормозит представление всей спецификации.

Эксперты в конкретных областях могут обмениваться своими мнениями с помощью электронных списков рассылки, имея возможность без каких-либо затруднений обсуждать то или иное средство, не тратя время на лишние разговоры. Однако в среде специалистов при ссылках на спецификацию HTML5 часто подразумевают ее первоначальный вариант, включающий, в частности, Geolocation API. Поэтому воспринимайте термин "HTML5" как "зонтичный", т.е. такой, который охватывает как основной набор элементов разметки, так и целый ряд новых программных интерфейсов с впечатляющими возможностями. На момент написания данной книги спецификация HTML5 состояла из следующих частей:

- элемент Canvas (2D и 3D);
- передача сообщений между документами;
- спецификация Geolocation API;
- MathML;
- Microdata;
- события, генерируемые сервером;
- Scalable Vector Graphics (SVG);
- программный интерфейс и протокол WebSocket;
- концепция веб-источника;

- веб-хранилища;
- спецификация Web SQL Database;
- потоки Web Workers;
- XMLHttpRequest Level 2.

Как нетрудно заметить, многие из программных интерфейсов, рассматриваемых в данной книге, входят в этот список. Каковы были принципы отбора спецификаций для нашего рассмотрения? Мы выбрали те из них, которые можно считать в той или иной мере близкими к завершению. Переносимость? В той или иной степени все эти интерфейсы доступны в ряде браузеров. Остальные (менее готовые) на данном этапе либо могут работать в какой-то одной конкретной версии браузера, либо пока что существуют лишь в виде идей.

В данной книге представлен современный (на момент написания книги) обзор предоставляемой браузерами поддержки, доступной для каждого из обсуждаемых нами средств HTML5. Однако, о чем бы мы вам ни рассказали, со временем это также потребует уточнения, поскольку объект нашего рассмотрения весьма изменчив. Вместе с тем, не стоит зря беспокоиться. Существуют великолепные сетевые ресурсы, с помощью которых вы сможете регулярно проверять текущее (а также будущее) состояние браузерной поддержки. На сайте `http://www.caniuse.com/` предоставляется исчерпывающий перечень доступных возможностей с указанием версий различных браузеров, в которых они поддерживаются, а на сайте `http://www.html5-test.com/` вы можете протестировать, какие средства HTML5 поддерживаются вашим браузером.

Заметим, что в книге не обсуждаются всевозможные обходные пути, основанные на эмуляции тех или иных средств, используя которые, можно добиться выполнения приложений HTML5 на устаревших браузерах. Вместо этого мы сосредоточимся главным образом на спецификации HTML5 и способах ее использования. Для каждого программного интерфейса будут приведены примеры кода, позволяющие проверить его доступность. Вместо того чтобы определять тип пользовательского агента, что не всегда можно сделать достаточно надежно, мы будем использовать обнаружение *самих средств*. Для этой цели также можно использовать *Modernizr* — библиотеку JavaScript, предоставляющую улучшенные возможности обнаружения средств HTML5 и CSS3. Мы настоятельно рекомендуем использовать библиотеку Modernizr в своих приложениях, поскольку в ней вы найдете наилучший инструментарий, предназначенный для этой цели.

Еще об HTML

Говорит Фрэнк: "Привет, меня зовут Фрэнк, мое хобби — рисование.

Одна из первых демонстраций возможностей элемента `canvas`, которую мне довелось увидеть, представляла собой простейшее графическое приложение, имитирующее пользовательский интерфейс Microsoft Paint. И хотя по своему техническому уровню приложение на десятки лет отставало от современных цифровых графических программ и (по состоянию на тот момент) могло запускаться лишь на некоторых из существующих в то время браузеров, его пример заставил меня всерьез задуматься над тем, какие широкие возможности за ним скрываются.

Как правило, когда у меня возникает необходимость в создании цифровых графических изображений, я использую настольные программы, установленные на локальном компьютере. Хотя некоторые из этих программ действительно великолепны, им не хватает тех возможностей, благодаря которым веб-приложения завоевали необычайную популярность. Проще говоря, они отсоединены от внешнего мира. До сих пор совместное использование цифровых изображений предполагало их экспорт из графического приложения и выгрузку в Интернет. Об организации совместной работы над рисунком или внесении правок на "живом" холсте не могло быть и речи. Приложения HTML5 позволяют сократить экспортный цикл и перенести в сетевую среду весь творческий процесс, а не только готовые изображения.

Круг приложений, которые не могут быть реализованы в HTML5, непрерывно сужается. Если говорить об обработке текста, то Интернет уже является двухсторонней коммуникационной средой для обмена текстовой информацией. Доступны модификации приложений для обработки текстов, предназначенные специально для Интернета. Однако графические приложения, такие как программы для рисования, видеомонтажа или 3D-моделирования, только начинают появляться в этой области.

Сейчас мы можем создавать замечательное программное обеспечение, предназначенное для работы с изображениями, музыкой, фильмами и т.д. К тому же оно будет работать как в Интернете, так и автономно — универсальная платформа с необычайно широкими возможностями, для которой главной "ареной действий" является Интернет."

Что нового в HTML5

Прежде чем приступить к собственно программированию, вкратце познакомимся с тем, что нового содержится в HTML5.

Новый дескриптор DOCTYPE и новое объявление кодировки символов

Во-первых, в соответствии с третьим принципом проектирования — *упрощение* — дескриптор DOCTYPE для веб-страниц значительно упростился. Взгляните, к примеру, на следующий дескриптор DOCTYPE в HTML 4:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
```

Сможет ли кто-нибудь это запомнить? Во всяком случае, мы с вами — вряд ли. Такие длинные директивы мы просто копируем в буфер обмена, а затем вставляем в нужное место на странице. Но при этом нас всегда гложут сомнения: "А правильно ли я выбрал тип документа?" В HTML5 эта проблема решается чрезвычайно просто:

```
<!DOCTYPE html>
```

Вот эту директиву мы уже в состоянии запомнить. Подобно дескриптору DOCTYPE, заметно "похудело" и объявление кодировки символов. Раньше оно выглядело так:

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

а теперь — вот так:

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

Использование нового дескриптора DOCTYPE запускает браузер для отображения страниц в режиме соответствия стандартам. На рис. 1.1 отображается информация, которую вы увидите на своем экране, если откроете веб-страницу сайта этой книги в браузере Firefox и выберете пункты меню Инструменты⇨Информация о странице (Tools⇨Page Info). В этом примере страница визуализируется в режиме соответствия стандартам.

В случае использования нового дескриптора DOCTYPE браузер визуализирует страницу в *режиме соответствия стандартам* (Standards compliance mode). Как вам, наверное, известно, веб-страницы могут иметь различные режимы визуализации, такие как **Quirks**, **Almost Standards** и **Standards**. DOCTYPE указывает браузеру, в каком режиме должны визуализироваться страницы и какие правила должны использоваться для их валидации (проверки корректности). В *режиме совместимости* (Quirks Mode) браузер игнорирует несоответствия стандарту и визуализирует страницы, имитируя поведение старого браузерного движка. HTML5, с одной стороны, вводит новые элементы, а с другой — объявляет некото-

рые элементы устаревшими (более подробно об этом говорится в следующем разделе), поэтому страницы, содержащие устаревшие элементы, не будут корректными, т.е. будут распознаны как не соответствующие стандарту.

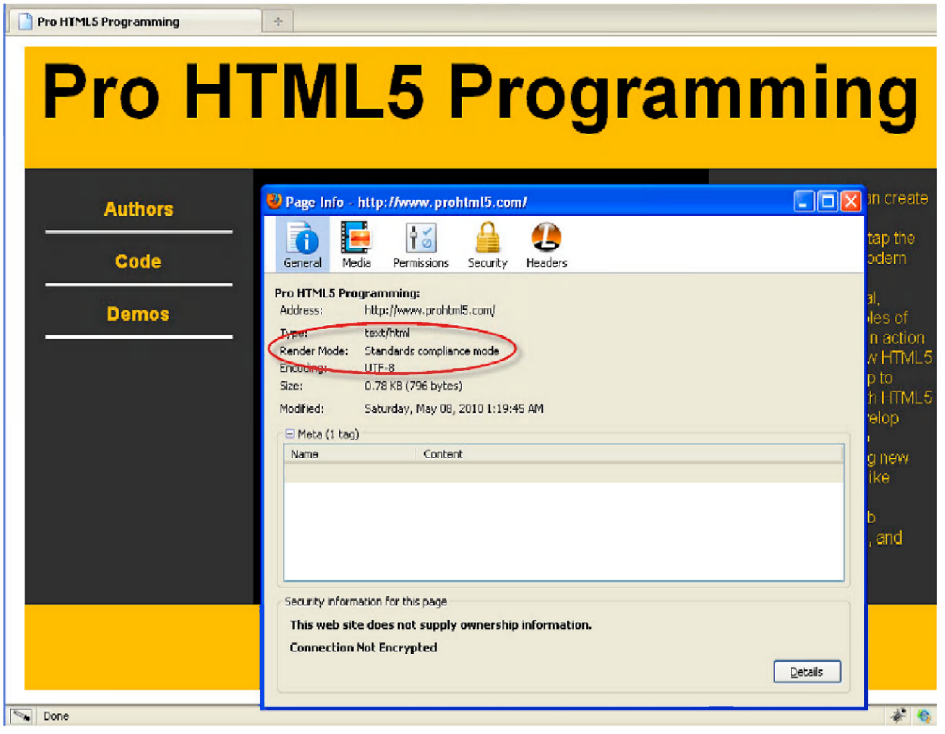


Рис. 1.1. Страница, визуализированная в режиме соответствия стандартам

Новые и устаревшие элементы

HTML5 вводит много новых элементов разметки, сгруппированных в соответствии с семью различными типами содержимого. Перечень типов содержимого приведен в табл. 1.1.

Таблица 1.1. Типы содержимого HTML5

Тип содержимого	Описание
Embedded	Содержимое, импортирующее в документ другие ресурсы, например `audio`, `video`, `canvas` или `iframe`
Flow	Элементы, используемые в теле документов и приложений, например `form`, `h1` или `small`
Heading	Заголовки разделов, например `h1`, `h2` или `hgroup`
Interactive	Содержимое, с которым взаимодействуют пользователи, например `audio`, `video`, `button` или `textarea`
Metadata	Элементы (обычно располагающиеся в разделе `head`), которые задают представление или поведение остальной части документа, например `script`, `style` или `title`
Phrasing	Текст и элементы разметки текста, например `mark`, `kbd`, `sub` или `sup`
Sectioning	Элементы, определяющие логические разделы документа, например `article`, `aside` или `title`

Ко всем вышеперечисленным элементам могут применяться стили CSS. Кроме того, некоторые из элементов, такие как `canvas`, `audio` и `video`, могут использоваться сами по себе, хотя для них и предусмотрены программные интерфейсы, обеспечивающие встроенные возможности тонкой программной настройки. Далее эти интерфейсы будут рассмотрены более подробно.

Обсуждение всех новых элементов выходит за рамки книги, однако элементы группы **Sectioning**, которые рассматриваются в следующем разделе, являются новыми. Элементы `canvas`, `audio` и `video` также впервые появились в HTML5.

Аналогичным образом, мы не собираемся приводить исчерпывающий перечень всех дескрипторов, которые считаются устаревшими (в Интернете вы найдете множество источников информации по этому вопросу). Отметим только, что многие элементы строковых (inline) описаний стилей, например `big`, `center`, `font` или `basefont`, исключены из спецификации, и вместо них следует использовать средства CSS.

Семантическая разметка

Один из типов содержимого, насчитывающий много новых элементов HTML5, отвечает за разбивку документа на отдельные части. В HTML5 для описания содержимого элементов определена новая, *семантическая* разметка. Использование семантической разметки не предоставляет никаких мгновенных преимуществ, но значительно облегчает верстку HTML-страниц, и, безусловно, в будущем поисковые системы воспользуются преимуществами этих элементов при организации поиска и индексирования страниц.

Как уже отмечалось, при разработке HTML5 сделана попытка максимально учесть предыдущий опыт использования HTML. Компания Google проанализировала миллионы страниц в поиске общих идентификаторов имен в дескрипторах `DIV` и обнаружила огромное количество повторений. Например, для разметки нижнего колонтитула страниц многие разработчики используют в элементе `DIV` атрибут `id="footer"`. Для создания различных частей документа HTML5 предоставляет новый набор элементов, который в современных браузерах можно использовать уже сегодня. Различные элементы семантической разметки приведены в табл. 1.2.

Таблица 1.2. Новые элементы секционирования документа в HTML5

Элемент секционирования	Описание
`header`	Содержимое верхней (начальной) части страницы или ее секции
`footer`	Содержимое нижней (завершающей) части страницы или ее секции
`section`	Секция веб-страницы
`article`	Содержимое внешней статьи
`aside`	Связанное содержимое или выдержка
`nav`	Навигационные ссылки

Ко всем перечисленным элементам могут применяться стили CSS. Фактически, как уже отмечалось, когда мы рассматривали применение принципа полезности при проектировании веб-страниц, HTML5 поощряет разделение представления и содержимого, поэтому для стилевого оформления страниц HTML5 следует всегда использовать CSS.

Пример разметки страницы HTML5 показан в листинге 1.1. Здесь используются новые объявления типа документа и кодировки, а также новые семантические элементы гипертекстовой разметки — проще говоря, новое содержимое, обеспечивающее разбиение документа на различные разделы. Файл с этим кодом находится в папке `code/intro`.

Листинг 1.1. Пример страницы HTML5

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>HTML5</title>
  <link rel="stylesheet" href="html5.css">

</head>

<body>

   <header>
     <h1>Header</h1>
     <h2>Subtitle</h2>
     <h4>HTML5 Rocks!</h4>
   </header>

    <div id="container">

        <nav>
          <h3>Nav</h3>
          <a href="">Link 1</a>
          <a href="">Link 2</a>
          <a href="">Link 3</a>
          <a href="">Link 4</a>
          <a href="">Link 5</a>

        </nav>

        <section>
            <article>
                <header>
                    <h1>Article Header</h1>
                </header>
                <p>Lorem ipsum dolor HTML5 nunc aut nunquam sit
                    amet, consectetur adipiscing elit.
                    Vivamus at est eros, vel fringilla urna.</p>
                <p>Per inceptos himenaeos. Quisque feugiat, justo
                    at vehicula pellentesque, turpis lorem
                    dictum nunc.</p>
                <footer>
                    <h2>Article Footer</h2>
                </footer>
            </article>
            <article>
                <header>
                    <h1>Article Header</h1>
                </header>
                <p>HTML5: "Lorem ipsum dolor nunc aut nunquam sit
                            amet, consectetur adipiscing elit.
                            Vivamus at est eros, vel fringilla
                            urna Pellentesque odio</p>
                <footer>
                    <h2>Article Footer</h2>
                </footer>
```

```
            </article>
        </section>

        <aside>
            <h3>Aside</h3>
            <p>HTML5: "Lorem ipsum dolor nunc aut nunquam sit
                       amet, consectetur adipiscing elit.
                       Vivamus at est eros, vel fringilla
                       urna. Pellentesque odio rhoncus</p>
        </aside>
        <footer>
            <h2>Footer</h2>
        </footer>
    </div>
</body>
```

Без применения стилей страница выглядела бы довольно скучно. В листинге 1.1 приведен CSS-код, который может быть использован для стилевого оформления содержимого. Обратите внимание, что в таблице стилей используются новые элементы CSS3, например скругленные углы (`border-radius`) и поворотные преобразования (`transform: rotate();`). Следует отметить, что спецификация CSS3, как и сама спецификация HTML5, все еще находится в стадии разработки, и из нее выделяются отдельные модули вспомогательных спецификаций, что упрощает оперативное использование браузерами тех ее частей, которые более или менее близки к завершению (например, преобразования, анимация и переходы выделены во вспомогательные спецификации).

Чтобы избежать возникновения конфликтов имен при возможных последующих изменениях спецификации, экспериментальные средства CSS3 снабжаются префиксом в виде строки, идентифицирующей производителя браузера. В настоящее время для отображения скругленных углов, градиентов, теней и преобразований необходимо использовать в объявлениях такие префиксы, как `-moz-` (для браузеров Mozilla), `o-`(для браузеров Opera) и `-webkit-`(для WebKit-браузеров, таких как Safari и Chrome).

Листинг 1.2. CSS-файл для страницы HTML5

```
body {
    background-color:#CCCCCC;
    font-family:Geneva,Arial,Helvetica,sans-serif;
    margin: 0px auto;
    max-width:900px;
    border:solid;
    border-color:#FFFFFF;
}

header {
    background-color: #F47D31;
    display:block;
    color:#FFFFFF;
    text-align:center;
}

header h2 {
    margin: 0px;
}
```

```
h1 {
    font-size: 72px;
    margin: 0px;
}

h2 {
    font-size: 24px;
    margin: 0px;
    text-align:center;
    color: #F47D31;
}

h3 {
    font-size: 18px;
    margin: 0px;
    text-align:center;
    color: #F47D31;
}

h4 {
    color: #F47D31;
    background-color: #fff;
    -webkit-box-shadow: 2px 2px 20px #888;
    -webkit-transform: rotate(-45deg);
    -moz-box-shadow: 2px 2px 20px #888;
    -moz-transform: rotate(-45deg);
    position: absolute;
    padding: 0px 150px;
    top: 50px;
    left: -120px;
    text-align:center;

}

nav {
    display:block;
    width:25%;
    float:left;
}

nav a:link, nav a:visited {
    display: block;
    border-bottom: 3px solid #fff;
    padding: 10px;
    text-decoration: none;
    font-weight: bold;
    margin: 5px;
}

nav a:hover {
    color: white;
    background-color: #F47D31;
}

nav h3 {
    margin: 15px;
    color: white;
}
```

```
#container {
    background-color: #888;
}

section {
    display:block;
    width:50%;
    float:left;
}

article {
    background-color: #eee;
    display:block;
    margin: 10px;
    padding: 10px;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
}

article header {
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
    padding: 5px;

}

article footer {
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
    padding: 5px;
}

article h1 {
    font-size: 18px;
}


aside {
    display:block;
    width:25%;
    float:left;
}

aside h3 {
    margin: 15px;
    color: white;
}

aside p {
    margin: 15px;
    color: white;
    font-weight: bold;
    font-style: italic;
}
```

```
footer {
    clear: both;
    display: block;
    background-color: #F47D31;
    color:#FFFFFF;
    text-align:center;
    padding: 15px;
}

footer h2 {
    font-size: 14px;
    color: white;
}


/* links */
a {
    color: #F47D31;
}

a:hover {
    text-decoration: underline;
}
```

На рис. 1.2 показан пример страницы из листинга 1.1, к которой применены стили CSS (и некоторые стили CSS3). Имейте, однако, в виду, что такого понятия, как "типичная страница HTML5", не существует. На самом деле страница может быть любой, и новые дескрипторы используются в данном примере главным образом в иллюстративных целях.

Наконец, заметим, что у вас могло сложиться ложное впечатление, будто браузеры визуализируют содержимое так, как если бы они действительно понимали семантический смысл новых элементов. Истина же состоит в том, что если бы мы назвали эти элементы как-то иначе, например `foo` и `bar`, но применили к ним те же стили, то они были бы визуализированы точно так же (хотя, безусловно, при этом были бы потеряны все преимущества в плане оптимизации работы поисковых систем). Единственным исключением является браузер Internet Explorer, который требует, чтобы элементы были частью DOM. Поэтому, если вы хотите увидеть эти элементы в Internet Explorer, их следует программным путем вставить в DOM и отобразить в виде блочных элементов.

Упрощение выбора элементов за счет использования селекторных функций

HTML5 вводит не только новые семантические элементы, но и новые упрощенные способы поиска элементов в DOM-модели страниц. В табл. 1.3 приведены предыдущие версии методов объекта `document`, с помощью которых разработчики могли находить на странице нужные элементы.

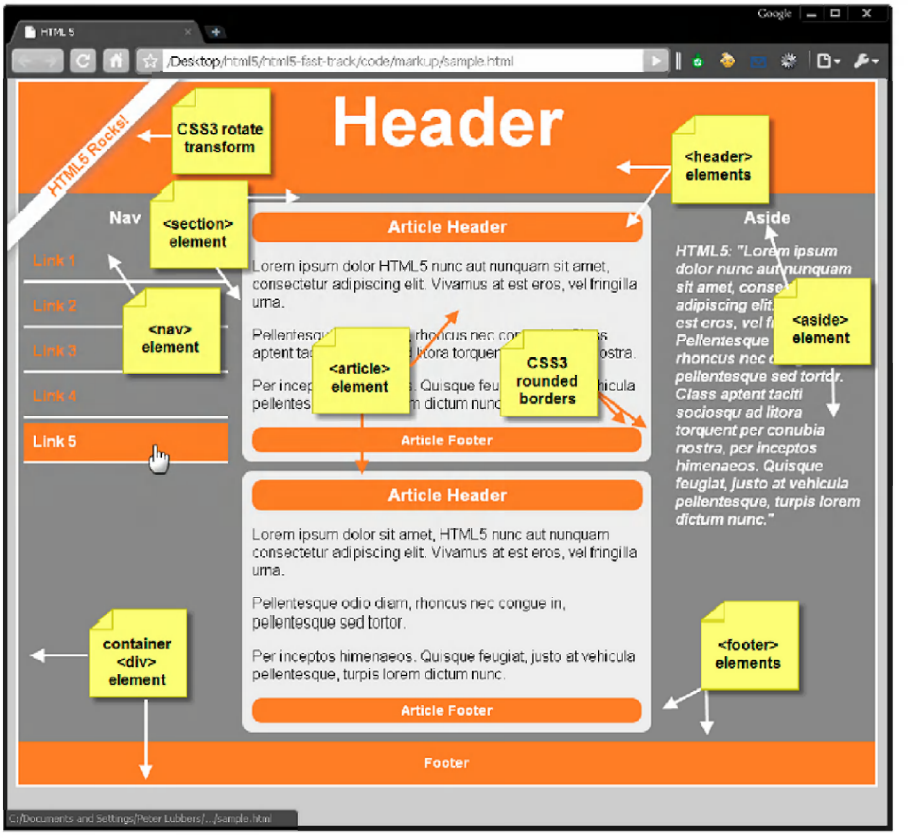


Рис. 1.2. Страница HTML5 с новыми семантическими элементами разметки

Таблица 1.3. Предыдущие методы JavaScript, предназначенные для поиска элементов

Метод	Описание	Пример
`getElementById()`	Возвращает элемент с указанным значением атрибута `id`	`<div id="foo"> getElementById("foo");`
`getElementsByName()`	Возвращает все элементы с атрибутами `name`, имеющими указанное значение	`<input type="text" name="foo"> getElementsByName("foo");`
`getElementsByTagName()`	Возвращает все элементы, имя дескриптора которых соответствует указанному значению	`<input type="text"> getElementsByTagName("input");`

Новые селекторные функции предлагают более точные способы указания искомых элементов, не требующие применения циклов и итеративного просмотра документа с помощью стандартной DOM-модели. В качестве возможностей поиска одного или нескольких элементов на странице HTML5 предоставляет те же селекторы, что и CSS. Например, в CSS уже имеются удобные селекторы для выбора элементов на основании признаков их вложенности, родства или наследования. В самых последних версиях добавлена поддержка дополнительных псевдоклассов (активизирован ли объект, отключен или отмечен) и почти любой комбинации их свойств и иерархических отношений, какую только можно себе представить. Для

выбора элементов в DOM-дереве с применением селекторов CSS просто используйте один из методов, представленных в табл. 1.4.

Таблица 1.4. Новые селекторные функции

Метод	Описание	Пример	Результат
`querySelector()`	Возвращает первый из встреченных на странице элементов, который соответствует указанному селектору (селекторам)	`querySelector ("input.error");`	Возвращает первый встреченный элемент ввода, имеющий класс `error`
`querySelector-All()`	Возвращает все элементы, соответствующие указанному селектору (селекторам)	`querySelectorAll ("#results td");`	Возвращает ячейки таблицы, находящиеся внутри элемента с идентификатором `id`, равным `results`

Селекторным функциям можно передавать сразу несколько селекторов, например:

```
// выбрать первый встреченный элемент документа с классом
// highClass или lowClass
var x = document.querySelector(".highClass", ".lowClass");
```

В случае функции `querySelector()` выбирается первый встреченный элемент, соответствующий любому из указанных селекторов. В случае же функции `querySelectorAll()` возвращаются все элементы, соответствующие любому из перечисленных селекторов. При наличии нескольких селекторов они разделяются запятыми.

Новые селекторные функции упрощают выбор разделов документа, что раньше представляло собой довольно трудную задачу. Предположим, например, что вы хотите иметь возможность выбора ячейки таблицы, над которой в данный момент находится указатель мыши. Это очень легко делается с помощью селекторов, как показано в листинге 1.3. Файл этого листинга также находится в папке `code/intro`.

Листинг 1.3. Использование селекторных функций

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="cp1251" />
  <title>Пример поиска с помощью селекторов
  </title>

  <style type="text/css">
    td {
      border-style: solid;
      border-width: 1px;
      font-size: 300%;
    }

    td:hover {
      background-color: cyan;
    }

    #hoverResult {
      color: green;
      font-size: 200%;
    }
  </style>
```

```
</head>

<body>
  <section>
    <!-- создать таблицу размером 3 × 3 -->
    <table>
      <tr>
        <td>A1</td> <td>A2</td> <td>A3</td>
      </tr>
      <tr>
        <td>B1</td> <td>B2</td> <td>B3</td>
      </tr>
      <tr>
        <td>C1</td> <td>C2</td> <td>C3</td>
      </tr>
    </table>

    <div>Установите фокус на кнопке, наведите указатель мыши на
 ячейку таблицы и нажмите клавишу <Enter> для идентификации ячейки
 путем вызова метода querySelector('td:hover').</div>
    <button type="button" id="findHover" autofocus>Найти объект
 поиска 'td:hover'</button>
    <div id="hoverResult"></div>

    <script type="text/javascript">
      document.getElementById("findHover").onclick = function() {
        // найти ячейку таблицы, над которой находится
        // указатель мыши
        var hovered = document.querySelector("td:hover");
        if (hovered)
           document.getElementById("hoverResult").innerHTML =
              hovered.innerHTML;
      }
    </script>
  </section>

</body>
</html>
```

Как видно из этого примера, для нахождения элемента, на который наведен указатель мыши, потребовалась всего одна строка кода:

```
var hovered = document.querySelector("td:hover");
```

Примечание

Селекторные функции не только более удобны, но зачастую и работают быстрее, чем средства обхода DOM для извлечения дочерних элементов с использованием традиционных библиотек. Для реализации быстро работающих таблиц стилей в браузерах используются чрезвычайно эффективные процедуры поиска с помощью селекторов.

В том, что в Консорциуме W3C отделили официальную спецификацию селекторов от спецификации CSS, нет ничего удивительного. Как вы уже видели, сфера применения селекторов не ограничивается стилевым оформлением документов. Подробное описание новых селекторов выходит за рамки книги, но если вы разработчик, ищущий оптимальные способы манипулирования элементами DOM, то но-

вые селекторные функции, обеспечивающие быструю навигацию по структуре приложения, несомненно, должны вас заинтересовать.

Протоколирование и отладка JavaScript-кода

Хотя с технической точки зрения инструменты протоколирования и отладки JavaScript-сценариев средствами браузера не имеют прямого отношения к HTML5, мы затронем эту тему, поскольку за последние несколько лет указанные средства значительно усовершенствовались. Среди замечательных инструментов, предназначенных для анализа веб-страниц и выполнения кода, первым было расширение браузера Firefox, получившее название Firebug.

Аналогичную функциональность теперь можно найти во встроенных инструментах разработки, предоставляемых другими браузерами: Веб-инспектор (Web Inspector) в Safari, Инструменты разработчика (Developer Tools) в Google Chrome, Средства разработчика (Developer Tools) в Internet Explorer и Dragon Fly в Opera. На рис. 1.3 показана панель Инструменты разработчика Google Chrome (открывается нажатием клавиш <Ctrl+Shift+J>), которая предоставляет развернутую информацию о веб-странице. Среди прочего инструментария в состав панели входят отладочная консоль (Console), средство просмотра ресурсов (Resources) и средство просмотра хранилищ данных (Storage).

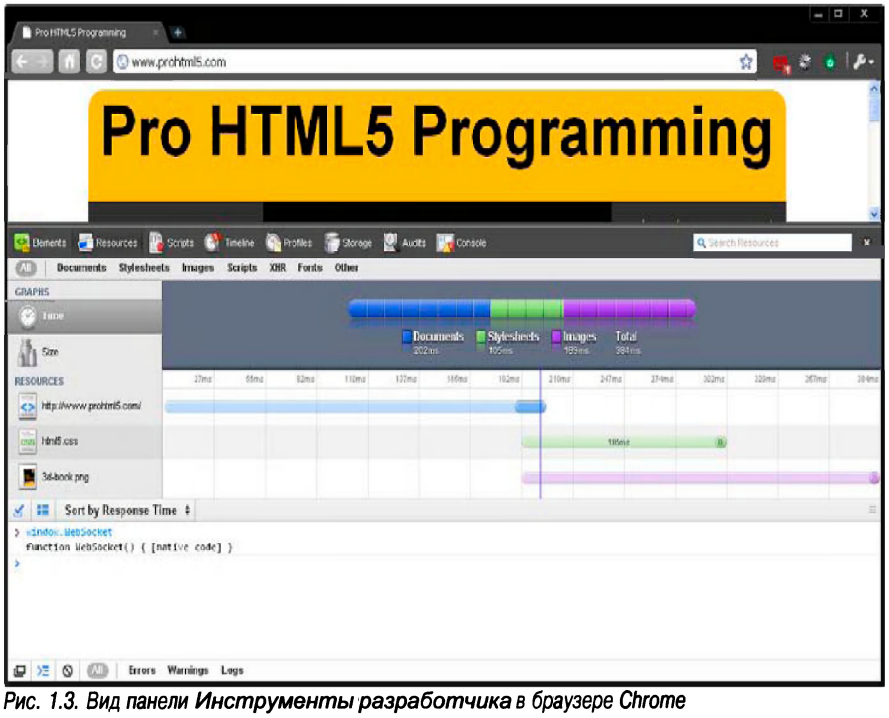


Рис. 1.3. Вид панели Инструменты разработчика в браузере Chrome

Многие средства разработки предлагают способы задания точек останова, позволяющих приостановить выполнение кода и проанализировать состояние программы и текущее состояние переменных. Функция `console.log` стала де-факто стандартом протоколирования для разработчиков JavaScript. Во многих браузерах предлагается представление сдвоенной панели, которое позволяет выводить сооб-

щения на консоль. Это гораздо удобнее, чем вызовы функции `alert()`, поскольку не сопровождается приостановкой выполнения программы.

window.JSON

JSON — это сравнительно новый, но быстро набирающий популярности способ представления данных. JSON является подмножеством синтаксиса JavaScript, в котором данные представляются в виде объектных литералов. Благодаря своей простоте, а также тому, что он естественным образом вписывается в концепцию JavaScript-программирования, формат JSON стал стандартом де-факто для обмена данными в приложениях HTML5. Каноническая библиотека для JSON включает в себя две функции: `parse()` и `stringify()` (что означает сериализовать, или преобразовать в строку).

Для использования JSON в старых браузерах необходимо установить библиотеку JavaScript (несколько таких библиотек можно найти на сайте `http://json.org/`). Операции синтаксического разбора и сериализации в JavaScript не всегда выполняются так быстро, как хотелось бы, и поэтому в новейших браузерах предусмотрены встроенные реализации JSON, которые можно вызывать из JavaScript. Встроенный объект JSON специфицируется как часть стандарта ECMAScript 5, охватывающего следующее поколение языка JavaScript. Это одна из первых частей стандарта ECMAScript 5, широкое внедрение которых не за горами. В каждом современном браузере имеется функция `window.JSON`, и можно надеяться, что вскоре мы станем свидетелями массового проникновения JSON в приложения HTML5.

DOM Level 3

Обработка событий всегда была одной из самых каверзных задач разработки веб-приложений. В отличие от большинства других браузеров, Internet Explorer не поддерживал стандартные библиотеки для событий и элементов. Ранее в Internet Explorer была реализована модель событий, отличающаяся от принятого впоследствии стандарта. Internet Explorer 9 (IE9) будет поддерживать средства DOM Level 2 и 3, так что в конечном счете вы сможете использовать один и тот же код для манипулирования объектами DOM и обработки событий во всех браузерах, поддерживающих спецификацию HTML5. В частности, это касается и никогда не теряющих своего значения методов `addEventListener()` и `dispatchEvent()`.

Monkey, SquirrelFish и прочие диковинки

Самые недавние новшества в браузерах никак не связаны ни с новыми тегами, ни с новыми программными интерфейсами. В последнее время одной из наиболее важных тенденций является быстрая эволюция движков JavaScript/ECMAScript в лидирующих браузерах. Подобно тому, как новые программные интерфейсы сделали возможным то, что до сих пор было невыполнимо в браузерах последних поколений, общее ускорение работы механизмов сценариев благотворно сказывается как на существующих веб-приложениях, так и на тех, в которых используются самые последние новинки HTML5. Думаете, ваш браузер не позволит обрабатывать сложные изображения или большие массивы данных, а также редактировать крупные документы? А если хорошенько подумать?

В последние годы среди производителей браузеров развязалась самая настоящая "гонка вооружений", в которой они пытаются выяснить между собой, чей движок JavaScript работает быстрее. В то время как ранние версии JavaScript были исключительно интерпретирующими, новейшие движки компилиру-

ют код сценария непосредственно в собственный машинный код, что на несколько порядков увеличивает производительность по сравнению с браузерами конца прошлого тысячелетия.

События начали стремительно развиваться с того момента, как Adobe передала свой движок оперативной (just-in-time, JIT) компиляции и виртуальную машину JavaScript (кодовое название Tamarin) проекту Mozilla в 2006 году. Хотя в самых последних версиях Mozilla сохранились лишь отдельные составляющие технологии Tamarin, передача этой технологии способствовала тому, что браузеры всех моделей теперь могут похвастаться новыми сценарными движками, имена которых интригуют не меньше, чем заявляемая производительность.

Таблица 1.5. Движки JavaScript различных браузеров

Браузер	Имя движка	Примечание
Apple Safari 5	Nitro (также известен как SquirrelFish Extreme)	Впервые появившийся в Safari 4 и доработанный в версии 5, вводит оптимизацию байт-кода и контекстно-потоковую компиляцию в собственный код
Google Chrome 5	V8	Начиная с Chrome 2, сбор мусора осуществляется с учетом предыстории (generational garbage collection), что обеспечивает высокую масштабируемость памяти без образования пауз в работе системы
Microsoft Internet Explorer 9	Chakra	Основное внимание уделено эффективной системе типов; демонстрирует десятикратное превосходство над IE8
Mozilla Firefox 4	JagerMonkey	Улучшенный по сравнению с версией 3.5 движок сочетает в себе высокую скорость интерпретации с компиляцией в собственный код в "горячих точках" деревьев трассировки (trace trees)
Opera 10.60	Carakan	Используется регистровый байт-код и выборочная компиляция в собственный код; заявленные характеристики браузера улучшены по сравнению с версией 10.50 на 75%

В целом, благодаря этой здоровой конкуренции между производителями браузеров, производительность JavaScript-сценариев как никогда ранее приблизилась к производительности собственного кода настольных приложений.

И снова об HTML

Говорит Питер: "Коль скоро речь зашла о конкуренции и скорости, хочу подключиться и я к этому разговору. Представлюсь — меня зовут Питер, и бег (особенно супермарафон) — мое страстное увлечение.

Бег на сверхдлинные дистанции — замечательный спорт, занятие которым дает возможность познакомиться с замечательными людьми[4]. Когда пробегаешь последние мили в супермарафоне, то по-настоящему узнаешь людей, хотя и в довольно необычной обстановке. В такие моменты в каждом человеке раскрывается истинная глубина его души, в которой всегда найдется место для замечательных друзей. Безусловно, забег — это соревнование, но прежде всего здесь господствует дух товарищества. Однако я, кажется, немного отвлекся от темы.

Так вот, следить за успехами своих друзей в забегах, в которых я по тем или иным причинам не могу принять участия (например, из-за того что пишу книгу по HTML5!), мне помогают специализированные веб-сайты, хотя — и в этом нет ничего удивительного — возможность наблюдать за участниками соревнований в режиме реального времени предлагается очень редко.

Несколько лет назад я случайно забрел на сайт одного европейского забега, в котором все было сделано так, как надо. Организаторы снабжали лидеров забега GPS-передатчиками и отображали их местоположение на карте (аналогичные примеры будут построены в этой книге с помощью программных интер-

[4] Если вас заинтересовала тема *супермарафонов*, можете прочитать об этом виде спорта и всех его разновидностях, которые приобретают в мире все большую популярность, в Википедии по адресу `http://en.wikipedia.org/wiki/Ultramarathon`.. — *Примеч. ред.*

фейсов Geolocation и WebSocket). И хотя сама реализация идеи была довольно примитивной, я почувствовал, что за всем этим скрывается огромный потенциал.

Сегодня, спустя всего лишь несколько лет, HTML5 предоставляет возможность создания подобных вебсайтов с использованием таких программных интерфейсов, как Geolocation, с помощью которого приложение может получать информацию о текущем местонахождении объекта, или WebSocket, который обеспечивает обновление геолокационной информации в режиме реального времени. Лично я не сомневаюсь — HTML5 первым пересек финишную черту и выиграл забег!"

Резюме

В этой главе были даны основные сведения об HTML5.

Вы познакомились с предпосылками появления этого языка, наиболее важными этапами его становления и узнали о четырех новых принципах разработки спецификаций, действующих в условиях зарождения новой эры HTML5: совместимость, удобство в использовании, независимость от типа браузера и универсальность доступа. Каждый из этих принципов открывает двери в мир новых возможностей, где нет места для многих прежних подходов и соглашений, которые к этому времени уже устарели. Также была представлена совершенно новая парадигма HTML5, не признающая подключаемых модулей, и дан ответ на вопрос, интересующий всех: что именно входит в спецификацию HTML5, а что не входит? Мы рассказали о том, что нового предлагает HTML5, упомянув, в частности, о новых объявлениях типа документа и кодировки, а также о новых элементах гипертекстовой разметки, и обратили внимание на обострение конкурентной борьбы за господство в сегменте JavaScript.

В следующей главе мы займемся программной стороной HTML5, начав с элемента Canvas.

Глава 2

Элемент Canvas

В этой главе исследуются возможности HTML5 Canvas API — мощного программного интерфейса, позволяющего генерировать и визуализировать графику, диаграммы, статические изображения и анимацию. Вы научитесь использовать основные функции визуализации для создания рисунков, способных масштабироваться и адаптироваться к среде браузера. Также будет описана методика создания динамических изображений, формируемых пользователем с помощью элемента ввода, реализованного в виде теплокарты. И конечно же, мы предупредим вас обо всех трудностях, с которыми вы можете столкнуться в процессе использования программного интерфейса HTML5 Canvas, и поделимся секретами их преодоления.

Для понимания материала, изложенного в этой главе, вполне хватит даже самого небольшого опыта работы с графикой, поэтому ничего не бойтесь и смело приступайте к освоению одного из наиболее мощных средств HTML.

Обзор средств HTML5 Canvas

Об использовании программного интерфейса HTML5 Canvas можно было бы написать целую книгу (причем весьма внушительного объема). Поскольку в нашем распоряжении имеется всего одна глава, далее обсуждаются лишь те функции, которые (по нашему мнению) используются наиболее часто.

Предыстория

Идея элемента `canvas` (холст) была впервые выдвинута и реализована компанией Apple в продукте Mac OS X WebKit для создания мини-приложений — так называемых *виджетов*. До появления этого элемента использование функций рисования в браузерах было возможно лишь при условии привлечения подключаемых модулей, таких как Flash-плагины Adobe, языка векторной разметки (VML; только в Internet Explorer) или же нестандартных программных решений на основе JavaScript.

Попробуйте, например, нарисовать обычную наклонную линию без использования элемента `canvas`. Несмотря на кажущуюся простоту этой задачи, без доступа к библиотеке функций для рисования 2D-графики сделать это довольно трудно. Именно такую библиотеку и предоставляет элемент `canvas` в HTML5, а поскольку иметь такую библиотеку в браузере чрезвычайно полезно, этот элемент был включен в спецификацию HTML5.

Ранее компания Apple намекала на возможность того, что она сохранит за собой авторские права на рабочий проект спецификации `canvas`, подготовленный группой WHATWG, что несколько обеспокоило некоторых сторонников веб-стандартизации. Однако Apple в конечном счете раскрыла свои патенты на предложенных консорциумом W3C условиях патентного лицензирования без сохранения авторских прав.

Сравнение возможностей формата SVG и элемента `canvas`

Говорит Питер: "Существенно то, что `canvas` — исключительно *растровый* холст, и поэтому нарисованные на нем изображения, в отличие от векторных изображений в формате SVG (Scalable Vector Graphics), невозможно масштабировать без потери качества. Кроме того, созданные на холсте изображения не являются частью DOM-модели страницы или какого-либо пространства имен, что иногда рассматривается как их недостаток. С другой стороны, SVG-изображения свободно масштабируются при различных величинах разрешения и позволяют точно определять, в каком месте изображения был выполнен щелчок.

Почему же тогда спецификация WHATWG HTML5 не использует исключительно SVG? Несмотря на очевидные недостатки HTML5 Canvas, в пользу этого программного интерфейса говорят следующие два факта: Элемент `canvas` обеспечивает хорошую производительность в силу того, что не должен хранить объекты для каждого рисуемого примитива и сравнительно легко реализуется на основе многих популярных библиотек для создания 2D-графики, которые существуют в других языках программирования. В конце концов, лучше синица в руке, чем журавль в небе."

Что такое холст

Когда вы используете элемент `canvas` на своей странице, он создает на ней прямоугольную область. По умолчанию ширина этой области составляет 300 пикселей, а высота — 150, но при желании для элемента `canvas` можно задать другие размеры и указать другие атрибуты. В листинге 2.1 представлен простейший элемент `canvas`, который можно добавить на HTML-страницу.

Листинг 2.1. Базовый элемент `canvas`

```
<canvas></canvas>
```

Включенным в страницу элементом `canvas` можно как угодно манипулировать, используя JavaScript. В него можно добавлять графику, линии и текст, вы можете сами рисовать на нем и даже добавлять в него полноценную анимацию.

Программный интерфейс HTML5 Canvas поддерживает те же операции 2D-рисования, что и большинство современных операционных систем и фреймворков. Если на протяжении последних лет вам приходилось заниматься программированием 2D-графики, то, работая с HTML5 Canvas, вы, скорее всего, будете чувствовать себя уверенно, поскольку спецификация специально спроектирована так, чтобы напоминать существующие системы. Если же такого опыта у вас нет, то вам предстоит узнать, насколько более мощной может быть система визуализации по сравнению с теми трюками с изображениями и CSS, которые разработчики годами использовали для создания веб-графики.

Чтобы программным путем использовать холст, прежде всего необходимо получить его *контекст* (context). После этого можно выполнить необходимые действия по отношению к контексту, а затем окончательно применить к нему их результат. Выполнение изменений на холсте аналогично выполнению транзакций в базах данных: сначала вы подготавливаете некоторые действия, а затем финализируете транзакцию.

Координаты холста

Как показано на рис. 2.1, координаты на холсте отсчитываются от точки с координатами `x=0, y=0`, находящейся в левом верхнем углу, которую мы будем называть *началом координат*, или *началом отсчета*. Координаты увеличиваются в горизонтальном направлении вдоль оси **x** и вертикальном — вдоль оси **y** (численные значения координат выражаются в пикселях).

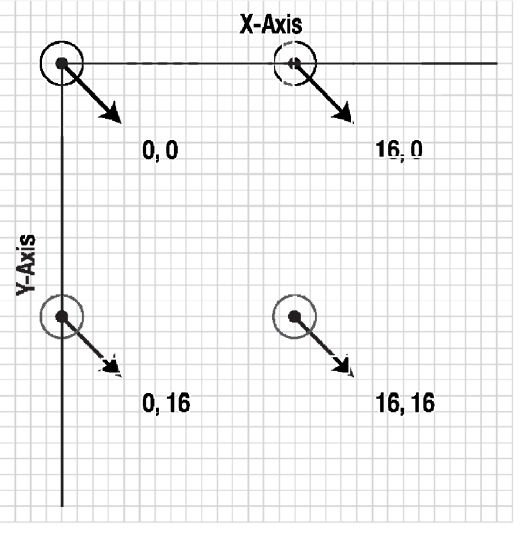


Рис. 2.1. Расположение начала отсчета координат x и y на холсте

Когда не следует использовать элемент `canvas`

Как бы ни был замечателен и полезен элемент `canvas`, его *не следует* использовать, если для решения задачи вполне подойдут другие элементы. Например, нельзя назвать хорошей идеей динамическое рисование всех заголовков HTML-документа на холсте, поскольку для этого достаточно использовать стили заголовков (H1, H2 и т.д.), специально предназначенные для этой цели.

Альтернативное содержимое

Если доступ к вашей веб-странице осуществляется браузером, который не поддерживает элемент `canvas` или некоторое подмножество средств HTML5 Canvas, то не помешает предоставить альтернативное содержимое (более подробно о поддержке браузерами элемента `canvas` см. табл. 2.1). Например, можно предоставить альтернативное изображение или некоторый текст, поясняющий, что именно смог бы увидеть пользователь на этом месте, если бы использовал современный браузер. В листинге 2.2 показано, как можно задать альтернативный (замещающий) текст внутри элемента `canvas`. Браузеры, не поддерживающие элемент `canvas`, могут просто выводить это альтернативное содержимое.

Листинг 2.2. Использование замещающего текста в элементе `canvas`

```
<canvas>
  Для отображения элемента canvas обновите браузер!
</canvas>
```

Вместо текста можно также указать изображение, которое будет выводиться, если поддержка элемента `canvas` в браузере отсутствует.

А что можно сказать относительно средств элемента `canvas`, предназначенных для людей с ограниченными возможностями?

Говорит Питер: "В связи с предоставлением альтернативных изображений или текста возникают вопросы доступности альтернативных средств для людей с *ограниченными возможностями* — область, в которой спецификация HTML5 Canvas, к сожалению, еще страдает заметными недостатками. Например, не существует собственного метода, обеспечивающего вставку в элемент `canvas` альтернативного текста вместо изображений, равно как не существует и метода, который предоставлял бы альтернативный текст, совпадающий с текстом, генерируемым с помощью текстовых функций Canvas API. На момент написания книги удачного решения проблем доступности средств элемента `canvas` для людей с ограниченными возможностями в случае динамически генерируемого содержимого не существовало, но над этим усиленно работает специально созданная группа. Будем надеяться, что со временем ситуация исправится."

CSS и элемент `canvas`

Как и в случае большинства элементов HTML, для добавления рамок, пробелов, полей и т.п. в элемент `canvas` можно использовать CSS-стили. Кроме того, некоторые атрибуты CSS наследуются содержимым `canvas`. Хорошим примером этого могут служить шрифты, поскольку по умолчанию к шрифтам, рисуемым на холсте, применяются настройки, используемые для самого элемента `canvas`.

Свойства, устанавливаемые для контекста, используемого в операциях с элементом `canvas`, подчиняются синтаксису, который, возможно, уже знаком вам по опыту работы с CSS. Например, для задания цветов и шрифтов в контексте используется та же нотация, что и в любом другом HTML- или CSS-документе.

Поддержка спецификации HTML5 Canvas браузерами

В настоящее время поддержка спецификации HTML5 Canvas предоставляется во всех браузерах, кроме Internet Explorer. Однако имеются некоторые части спецификации, добавленные позже, которые не имеют столь широкой поддержки. В качестве примера можно привести текстовые функции Canvas API. Однако в целом разработка спецификации продвинулась достаточно далеко, и вероятность внесения в нее существенных изменений невелика. Как следует из табл. 2.1, на момент написания книги спецификация HTML5 Canvas уже поддерживалась многими браузерами.

Таблица 2.1. Поддержка спецификации HTML5 Canvas различными браузерами

Браузер	Поддержка
Chrome	Поддерживается в версии 1.0 и выше
Firefox	Поддерживается в версии 1.5 и выше
Internet Explorer	Не поддерживается
Opera	Поддерживается в версии 9.0 и выше
Safari	Поддерживается в версии 1.3 и выше

Приведенные данные свидетельствуют о том, что спецификация HTML5 Canvas поддерживается всеми браузерами, за исключением Internet Explorer. Если вы хотите использовать элементы `canvas` в Internet Explorer, воспользуйтесь открытым проектом **explorercanvas** (`http://code.google.com/p/explorercanvas`). Для этого необходимо просто убедиться в том, что текущим браузером является Internet Explorer, и включить дескриптор `script` в свои веб-страницы для запуска сценария, как показано ниже.

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

Универсальная поддержка спецификации Canvas крайне желательна для разработчиков, и поэтому постоянно возникают новые проекты, целью которых является добавление этой поддержки в среду устаревших и нестандартных браузеров. Компания Microsoft заявила о поддержке программного интерфейса Canvas в Internet Explorer 9, и поэтому поддержка данной спецификации всеми основными типами веб-браузеров уже не за горами.

В силу неоднородности этой поддержки целесообразно сначала протестировать, поддерживается ли программный интерфейс HTML5 Canvas данным браузером, и только после этого использовать его. Далее будет показано, как организовать такую проверку программным путем.

Программный интерфейс HTML5 Canvas

Этот раздел посвящен подробному рассмотрению возможностей программного интерфейса HTML5 Canvas. Эти возможности будут продемонстрированы на примере создания незамысловатого логотипа в виде изображения участка леса с несколькими деревьями и удобной тропинкой, по которой вполне может проходить маршрут соревнований по марафону на пересеченной местности. Хотя наш пример и не может претендовать на получение какой-либо награды на конкурсе работ по графическому дизайну, он достаточно хорошо иллюстрирует различные возможности программного интерфейса HTML5 Canvas.

Проверка поддержки в браузере

Прежде чем использовать элемент HTML5 `canvas`, следует проверить, поддерживает ли его браузер. Это позволит предоставить замещающий текст в том случае, если в устаревшем браузере пользователя такая поддержка отсутствует. Один из способов организации подобной проверки показан в листинге 2.3.

Листинг 2.3. Проверка поддержки элемента `canvas` в браузере

```
try {
  document.createElement("canvas").getContext("2d");
  document.getElementById("support").innerHTML =
    "Ваш браузер поддерживает элемент HTML5 Canvas.";
} catch (e) {
  document.getElementById("support").innerHTML = "Ваш браузер
     не поддерживает элемент HTML5 Canvas.";
}
```

В этом примере делается попытка создать объект `canvas` и получить доступ к его контексту. Если возникает ошибка, она перехватывается, и пользователь извещается о том, что элемент `canvas` не поддерживается данным браузером. Определенный ранее на странице элемент `support` обновляется помещением в него сообщения, информирующего о наличии или отсутствии указанной поддержки.

Этот тест позволяет судить лишь о состоянии поддержки элемента `canvas` как такового. Он ничего не говорит о том, какие именно возможности элемента поддерживаются. На момент написания книги данная спецификация была устойчива и надежно поддерживалась, так что беспокоиться по этому поводу, в общем-то, нечего.

Наконец, еще раз подчеркнем, что всегда неплохо предусмотреть для элемента canvas вывод альтернативного содержимого, аналогично тому, как это сделано в листинге 2.3.

Добавление элемента canvas на страницу

Чтобы добавить на веб-страницу элемент canvas, не требуется больших усилий. Как это сделать, показано в листинге 2.4.

Листинг 2.4. Элемент canvas

```
<canvas height="200" width="200"></canvas>
```

Результирующий холст будет отображаться на странице как "невидимый" прямоугольник с размерами 200×200 пикселей. Если хотите добавить окружающую холст рамку, используйте приведенный в листинге 2.5 код, создающий обычное обрамление средствами CSS.

Листинг 2.5. Элемент canvas, ограниченный сплошной рамкой

```
<canvas id="diagonal" style="border: 1px solid;"
  width="200" height="200">
</canvas>
```

Обратите внимание на включение атрибута ID со значением "diagonal", который упростит нахождение элемента canvas программным путем. Наличие подобного идентификатора весьма существенно для любого элемента canvas, поскольку любая полезная операция над этим элементом должна выполняться из сценария. Отсутствие атрибута ID значительно затруднит получение доступа к элементу, с которым необходимо взаимодействовать.

На рис. 2.2 иллюстрируется, как будет выглядеть в браузере элемент canvas, определенный в листинге 2.5.

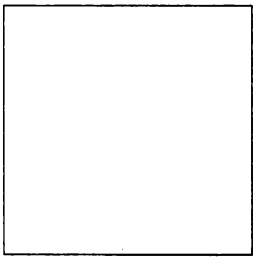


Рис. 2.2. Отображение простого элемента HTML5 canvas на веб-странице

Это не очень впечатляет, но любой художник скажет вам, что в этом рисунке скрыт огромный потенциал. Давайте же изобразим что-нибудь на этом девственно чистом холсте. Как уже отмечалось ранее, нарисовать на веб-странице наклонную линию, не используя возможности HTML5 Canvas, не так-то просто. Посмотрим, легко ли это сделать с помощью элемента canvas. Как видно из листинга 2.6, для прорисовки наклонной линии на холсте, который мы перед этим добавили в страницу, достаточно всего лишь нескольких строк кода.

Листинг 2.6. Вычерчивание наклонной линии на холсте

```
<script>
  function drawDiagonal() {
    // Получить элемент canvas и его графический контекст
    var canvas = document.getElementById('diagonal');
    var context = canvas.getContext('2d');

    // Создать путь в абсолютных координатах
    context.beginPath();
    context.moveTo(70, 140);
    context.lineTo(140, 70);

    // Прочертить линию на холсте
    context.stroke();
  }

   window.addEventListener("load", drawDiagonal, true);
</script>
```

Давайте проанализируем код JavaScript, использованный для построения наклонной линии. Несмотря на простоту этого примера, он охватывает все существенные аспекты работы с программным интерфейсом HTML5 Canvas.

Сначала осуществляется доступ к объекту `canvas`, для чего используется ссылка на значение идентификатора ID. В данном примере значением идентификатора является `"diagonal"`. Далее создается переменная `context` и вызывается метод `getContext()`объекта `canvas`, которому передается тип требуемого контекста. В данном случае передается строка `"2d"`, что означает двухмерный контекст, единственно доступный в настоящее время.

Примечание

Поддержка трехмерного (3D) контекста может появиться в одной из будущих версий спецификации.

Полученный контекст используется для выполнения операций рисования. В данном случае вычерчивается наклонная линия, что достигается вызовом трех методов, а именно `beginPath()`, `moveTo()` и `lineTo()`, с передачей координат начала и конца отрезка в качестве параметров.

Вызов методов `moveTo()` и `lineTo()` не приводит к фактическому завершению операции и прорисовке линии на холсте — для этого следует вызвать метод `context.stroke()`. Полученная наклонная линия изображена на рис. 2.3.

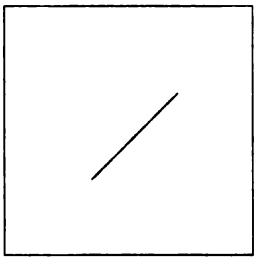


Рис. 2.3. Наклонная линия, нарисованная на холсте

Победа! Хотя усмотреть в этом результате некое предвестие революции довольно затруднительно, не следует забывать, что вычерчивание наклонной линии классическими методами HTML потребовало бы выполнения довольно сложных манипуляций, связанных с искажением изображений, а также использованием замысловатых CSS- и DOM-объектов и прочих элементов черной магии. Теперь обо всем этом можно позабыть.

Как видно из кода примера, все операции на холсте выполняются через объект контекста. То же самое можно сказать и в отношении остальных видов взаимодействия с холстом, поскольку доступ ко всем важным функциям, связанным с визуальным выводом, осуществляется лишь из контекста, а не из самого объекта `canvas`. Несмотря на то что в этой главе мы будем часто говорить о выполнении операций рисования на холсте, не забывайте: на самом деле это означает, что мы будем работать с объектом контекста, предоставляемым объектом `canvas`.

Последний пример демонстрирует, что многие операции, выполняемые над контекстом, не приводят к немедленному обновлению поверхности рисования. Выполнение таких функций, как `beginPath()`, `moveTo()` и `lineTo()`, не сопровождается мгновенным изменением внешнего вида холста. То же самое справедливо для многих функций, задающих стили, используемые на холсте, и устанавливающих предпочтительные значения параметров для него. *Путь* (path) становится видимым на экране лишь после того, как к нему будут применены методы `stroke()` или `fill()`. Иными словами, непосредственное обновление холста будет происходить лишь при выводе изображений или текста, а также при прорисовке, заполнении или очистке прямоугольников.

Использование преобразований в рисунках

Перейдем к рассмотрению другого способа рисования на холсте — *преобразованиям*. В следующем примере будет получен тот же результат, что и в предыдущем, но используемый для этого код будет другим. Если в данном случае, ввиду его простоты, еще и можно было бы возражать против привлечения преобразований на том основании, что это излишне усложняет решение, то при переходе к задачам, представляющим практический интерес, преобразования следует рассматривать как один из элементов оптимальной практики программирования, и поэтому мы будем интенсивно использовать их в оставшихся примерах. Без понимания преобразований невозможно в полной мере освоить все возможности программного интерфейса HTML5 Canvas.

Вероятно, наиболее простой способ понять, как работает система преобразований, — по крайней мере, самый простой из тех, которые не требуют привлечения устрашающих математических формул и совершения таинственных манипуляций, — это представить ее в виде модифицирующего слоя, расположенного между буфером посылаемых вами команд и выводом на экранный холст. Этот слой присутствует всегда, даже в тех случаях, когда вы не взаимодействуете с ним явно.

Модификации, или, в терминологии графических систем, *преобразования*, могут применяться последовательно, объединяться или видоизменяться по необходимости. Каждая графическая операция пропускается через модифицирующий слой, где графические объекты подвергаются всем необходимым изменениям, прежде чем выводятся на холст. И хотя общая картина при этом немного усложняется, возможности графической системы резко возрастают. Именно благодаря такому подходу стали возможными мощные преобразования, которые современное программное обеспечение для обработки изображений поддерживает в режиме реального времени, и это все в рамках библиотеки функций, сложной ровно настолько, насколько это абсолютно необходимо.

Не совершайте ошибку, полагая, что в случае отказа от использования преобразований в своем коде вы повысите его производительность. В реализации холста механизм визуализации всегда неявно использует преобразования, независимо от того, вызываете вы их непосредственно или не вызываете. Неплохо заранее представлять себе, как работает графическая система, поскольку эти знания окажутся для вас крайне важными при любой попытке выйти за рамки простейших графических операций.

Ключевая рекомендация по созданию повторно используемого кода состоит в том, что рисование, как правило, следует выполнять *с привязкой к началу координат*, а для придания рисунку окончательного вида и помещения его в нужное место на странице необходимо применять преобразования — масштабирование, трансляцию (перенос), повороты и т.п., как показано на рис. 2.4.

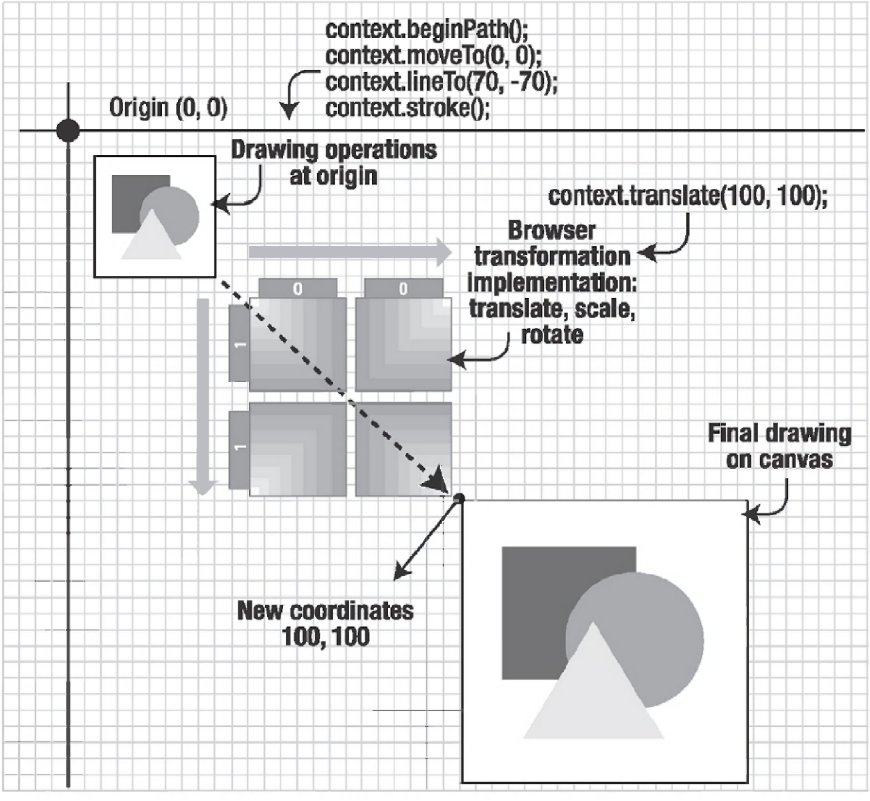


Рис. 2.4. Рисование с привязкой к началу координат и последующее преобразование результата

Применение этой рекомендации наглядно демонстрируется в листинге 2.7 на примере простейшего преобразования — `translate`.

Листинг 2.7. Создание наклонной линии на холсте с помощью трансляции

```
<script>
  function drawDiagonal() {
    var canvas = document.getElementById('diagonal');
    var context = canvas.getContext('2d');
```

```
    // Сохранить копию текущего состояния контекста
    context.save();

    // Транслировать графический контекст вправо и вниз
    context.translate(70, 140);

    // Нарисовать ту же линию, что и раньше, но с привязкой
    // к началу координат
    context.beginPath();
    context.moveTo(0, 0);
    context.lineTo(70, -70);
    context.stroke();

    // Восстановить прежнее состояние контекста
    context.restore();
  }

  window.addEventListener("load", drawDiagonal, true);
</script>
```

Давайте проанализируем сценарий JavaScript, использованный для создания второго варианта наклонной линии с помощью трансляции.

1. Прежде всего, мы получаем доступ к объекту `canvas`, ссылаясь на значение его свойства ID (в данном случае — `"diagonal"`).
2. Затем путем вызова метода `getContext()` объекта `canvas` получается значение переменной `context`.
3. Полученный контекст, который пока еще не подвергся никаким изменениям, необходимо сохранить, чтобы по завершении графических операций и преобразований можно было восстановить его исходное состояние. В противном случае внесенные изменения (трансляция, масштабирование и т.п.) будут применяться к контексту при выполнении всех будущих графический операций, что может быть нежелательным. Сохранение того состояния контекста, в котором он находился до выполнения преобразований, позволит впоследствии вновь обратиться к нему, если в этом возникнет необходимость.
4. На следующем этапе к контексту применяется трансляция. Это означает, что при визуализации любого рисунка к координатам, используемым операцией рисования (в данном случае — рисования наклонной линии), будут добавляться предоставленные вами координаты трансляции, тем самым перемещая линию в ее конечное положение, но только после того, как операция рисования будет завершена.
5. После применения трансляции контекста можно выполнить обычные графические операции по созданию наклонной линии. В данном случае эта линия создается путем вызова трех методов — `beginPath()`, `moveTo()` и `lineTo()`, которые в этот раз выполняются с привязкой к началу координат (`0,0`), а не к точке с координатами (`70,40`).
6. Сформированную таким способом линию можно визуализировать на холсте (например, путем прорисовки), вызвав для этого метод `context.stroke()`.
7. Наконец, мы восстанавливаем исходное состояние контекста для того, чтобы примененная к нему трансляция не модифицировала все последующие операции (рис. 2.5).

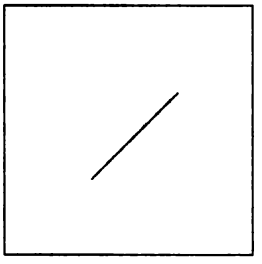


Рис. 2.5. Транслированная наклонная линия на холсте

Несмотря на полную тождественность новой линии с прежней, она была создана с использованием мощных возможностей преобразований, которые вы сможете по достоинству оценить по мере чтения оставшихся разделов главы.

Работа с путями

Можно было бы продолжить наше рассмотрение демонстрацией более ярких примеров рисования линий, однако вы уже достаточно подготовлены к тому, чтобы перейти к изучению более сложных объектов — *путей* (paths). В HTML5 Canvas путями представляется любая фигура, которую требуется визуализировать. Исходная линия из нашего примера представляла собой именно путь, на что довольно прозрачно указывает название метода `beginPath()`, который был вызван в самом начале выполнения операций рисования. Однако при необходимости могут быть созданы гораздо более сложные пути, состоящие из множества прямолинейных отрезков, криволинейных сегментов и даже подчиненных путей. Если вы хотите иметь возможность нарисовать на холсте практически любую фигуру, то вам стоит внимательно присмотреться к функциям для работы с путями.

Всякий раз, когда вы собираетесь изобразить геометрическую фигуру или путь, прежде всего необходимо вызвать метод `beginPath()`. Эта простая функция не получает никаких аргументов и лишь сигнализирует холсту, что вы приступаете к описанию новой фигуры. Главным образом, ее вызов необходим холсту, чтобы он мог вычислить внутренние и внешние границы создаваемой фигуры для последующего вычерчивания линий и заливки внутренних областей.

Путь постоянно отслеживает текущее местоположение, которым по умолчанию является начало координат. Текущая позиция указателя на холсте отслеживается его внутренними переменными, но вы будете изменять ее своими графическими программами.

Коль скоро был вызван метод `beginPath()`, можно приступить к вычерчиванию контуров нужной фигуры, применяя к холсту соответствующие функции. С простейшими функциями для построения путей в контексте вы уже знакомы:

- `moveTo(x,y)` — перемещает текущую позицию в новую точку с координатами `(x,y)` без прорисовки линии;
- `lineTo(x,y)` — перемещает текущую позицию в новую точку с координатами `(x,y)` и прорисовывает линию, соединяющую обе точки.

Образно говоря, эти вызовы различаются тем, что первый из них можно сравнить с поднятием рейсфедера и переносом его в другую точку на ватмане, тогда как второй равнозначен опусканию рейсфедера на бумагу и проведению прямой линии до новой точки. Однако, и на это необходимо обратить особое внимание, реально ничего не будет прочерчено до тех пор, пока к пути не будут применены методы

stroke() или fill(). Пока что мы всего лишь определили позиции в нашем пути, которые впоследствии будут использованы для "проявления" линий.

Следующей специальной функцией, воздействующей на путь, является closePath(). По своему поведению она очень похожа на функцию lineTo(), с тем лишь отличием, что в данном случае в качестве конечной точки будет автоматически использоваться начальная точка пути. Одновременно функция closePath() информирует холст о том, что текущая фигура замкнулась, т.е. образовала ограниченную со всех сторон область. Впоследствии это может пригодиться при прорисовке контуров и заливке фигуры.

Описанный процесс можно продолжать сколько угодно, создавая дополнительные подчиненные пути за счет добавления новых сегментов линий. В любой момент можно полностью очистить список пути и начать все с самого начала, вызвав функцию beginPath().

Как это всегда бывает в случае большинства сложных систем, лучше один раз увидеть, чем сто раз услышать. Оторвемся от наших "прямолинейных" примеров и используем программный интерфейс HTML5 Canvas для создания логотипа соревнований по бегу, одним из элементов которого будет изображение лесной тропинки. Подобно рисованию любой другой картины, мы начнем с создания основного элемента, роль которого в данном случае будет играть крона сосны. Используемый для этого код приведен в листинге 2.8.

Листинг 2.8. Функция, создающая путь для кроны дерева

```
function createCanopyPath(context) {
  // Создать путь, соответствующий кроне дерева
  context.beginPath();

  context.moveTo(-25, -50);
  context.lineTo(-10, -80);
  context.lineTo(-20, -80);
  context.lineTo(-5, -110);
  context.lineTo(-15, -110);

  // Верхушка дерева
  context.lineTo(0, -140);

  context.lineTo(15, -110);
  context.lineTo(5, -110);
  context.lineTo(20, -80);
  context.lineTo(10, -80);
  context.lineTo(25, -50);

  // Замкнуть путь на начальную точку
  context.closePath();
}
```

Как видно из этого листинга, здесь использованы те же команды перемещения из одной точки в другую и построения отрезка, что и в предыдущих примерах, только их стало больше. Основная часть кода формирует контур дерева, а завершающая — замыкает путь на начальную точку. В нижней части дерева явно чего-то недостает, но мы восполним этот пробел в последующих разделах, пририсовав к дереву ствол. Листинг 2.9 демонстрирует использование этой функции для фактического вывода рисунка на холст.

Листинг 2.9. Функция для рисования дерева на холсте

```
function drawTrails() {
  var canvas = document.getElementById('trails');
  var context = canvas.getContext('2d');

  context.save();
  context.translate(130, 250);

  // Создать путь, образующий крону
  createCanopyPath(context);

  // Прорисовать текущий путь
  context.stroke();
  context.restore();
}
```

Входящие в эту процедуру вызовы функций должны быть вам уже знакомы. Мы получаем контекст холста, сохраняем его для использования в будущем, изменяем местоположение текущей позиции, выполнив трансляцию, рисуем крону, а затем восстанавливаем ранее сохраненное состояние контекста. Результат нашего творчества — крона дерева, изображенная с помощью простых линий, — представлен на рис. 2.6. Впоследствии мы улучшим рисунок, но для первого шага это уже неплохо.

Рис. 2.6. Простой путь, представляющий крону дерева

Работа со стилями линий

Программный интерфейс HTML5 Canvas не был бы столь мощным и не обрел бы столь широкой популярности, если бы разработчики ограничивались созданием контурных рисунков, состоящих из простых линий, к тому же рисуемых одним только черным цветом. Чтобы придать кроне более похожий на дерево вид, воспользуемся возможностями стилизации линий. В листинге 2.10 приведены основные команды, позволяющие повысить привлекательность нарисованной фигуры за счет изменения свойств контекста.

Листинг 2.10. Использование стилизации линий

```
// Увеличить ширину линии
context.lineWidth = 4;
```

```
// Скруглить углы в местах сочленения путей
context.lineJoin = 'round';

// Изменить цвет линий
context.strokeStyle = '#663300';

// Наконец, нарисовать крону
context.stroke();
```

Добавляя указанные свойства до применения метода `stroke()`, мы изменяем внешний вид любой из рисуемых впоследствии фигур — по крайней мере, до тех пор пока не будет восстановлено исходное состояние контекста.

В этом примере мы сначала увеличиваем ширину линий до 4 пикселей. Далее мы устанавливаем для свойства `lineJoin` значение `round` (скругление), которому соответствует скругление углов в местах сочленения линейных сегментов фигуры. Мы могли бы также воспользоваться альтернативными вариантами обработки углов, установив для свойства `lineJoin` одно из двух других возможных значений: `bevel` (фаска) или `miter` (скос) (с соответствующей дополнительной настройкой свойства `context.miterLimit`).

Наконец, мы изменяем цвет линий, используя свойство `strokeStyle`. В данном случае мы указываем цвет, используя CSS-значения, но, как будет показано в следующих разделах, можно создавать более красочные картинки, устанавливая в качестве значений свойства `strokeStyle` шаблонные узоры или градиенты.

Кроме того, хотя здесь это и не применяется, можно было бы также задать стили окончания линий, установив для свойства `lineCap` одно из значений `butt`, `square` или `round`. Внешний вид несколько приукрашенной кроны, теперь уже нарисованной с помощью более толстых сглаженных линий коричневого цвета, проиллюстрирован на рис. 2.7.

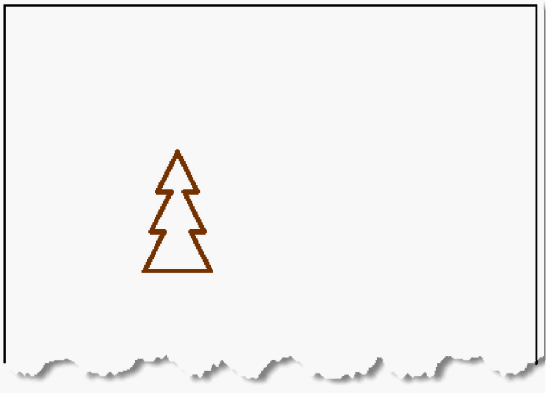


Рис. 2.7. Рисунок кроны, полученный с использованием стилизованных линий

Работа со стилями заливки

Как и можно было ожидать, стилизация линий — не единственный способ воздействия на внешний вид фигур, рисуемых на холсте. Другой широко используемый способ изменения внешнего вида фигур состоит в том, чтобы указать, каким именно образом должны закрашиваться внутренние области путей и подчиненных

путей. Листинг 2.11 показывает, как легко закрасить область, ограниченную контуром кроны, приятным зеленым цветом.

Листинг 2.11. Использование стиля заливки

```
// Установить для заливки зеленый цвет и закрасить крону
context.fillStyle = '#339900';
context.fill();
```

Сначала здесь с помощью свойства `context.fillStyle` устанавливается подходящий цвет заливки. Как будет показано далее, для заливки можно также использовать градиенты или шаблонные узоры. Далее нам остается лишь вызвать функцию `fill()` контекста, чтобы установить зеленый цвет для всех пикселей, заключенных внутри всех замкнутых путей, из которых состоит фигура, как показано на рис. 2.8.

Рис. 2.8. Закрашенная крона

Ввиду того, что обводка кроны была прорисована до ее заливки, заливка распространилась лишь на часть ширины пути. Это произошло по той причине, что линии, имеющие толщину (в данном случае толщина линий составляет 4 пикселя), центрируются вдоль обводки фигуры. Заливка применяется ко всем пикселям внутренней области фигуры, а раз так, она покрывает половину пикселей обводки. Если вы хотите, чтобы обводка отображалась с сохранением полной толщины, достаточно применить заливку *до* применения обводки.

Заливка прямоугольника

Любое дерево заслуживает прочного основания. К счастью, мы оставили место для ствола в пути, описывающем исходную фигуру. В листинге 2.12 показано, как проще всего визуализировать ствол дерева с помощью специальной функции `fillRect()`.

Листинг 2.12. Использование специальной функции `fillRect()`

```
// Изменить цвет заливки на коричневый
context.fillStyle = '#663300';

// Закрасить прямоугольник, представляющий ствол дерева
context.fillRect(-5, -50, 10, 50);
```

Здесь вновь был использован коричневый цвет. Однако, вместо того чтобы явно создавать прямоугольник ствола с помощью функции `lineTo()`, мы нарисовали весь ствол с помощью одной операции, используя функцию `fillRect()`. Эта функция принимает в качестве аргументов координаты `x` и `y` левого верхнего угла, а также ширину и высоту прямоугольника, и сразу же закрашивает его, используя текущий стиль заливки.

Для прямоугольников предусмотрены также специальные функции `strokeRect()` и `clearRect()`, которые в данном случае не используются. Первая из них предназначена для вычерчивания прямоугольника на основании координат его местоположения и размеров, а вторая очищает прямоугольную область от любого содержимого и восстанавливает ее исходный прозрачный цвет.

Анимация холста

Говорит Брайан: "Возможность очистки прямоугольной области холста имеет ключевое значение для создания анимации и игр с помощью программного интерфейса HTML5 Canvas. Многократно повторяя на отдельных участках холста цикл рисования и очистки, можно создать иллюзию анимации, чему уже есть множество примеров в Интернете. Однако для того чтобы обеспечить плавное воспроизведение анимации, необходимо использовать средства *отсечения* (clipping) и, возможно, даже вспомогательный буферный холст, позволяющий избежать мерцания изображений, вызванного частой очисткой холста. Хотя рассмотрение анимации выходит за рамки данной книги, ничто не мешает вам самостоятельно исследовать ее возможности."

Внешний вид созданного нами дерева, которое уже имеет и крону, и ствол, показан на рис. 2.9.

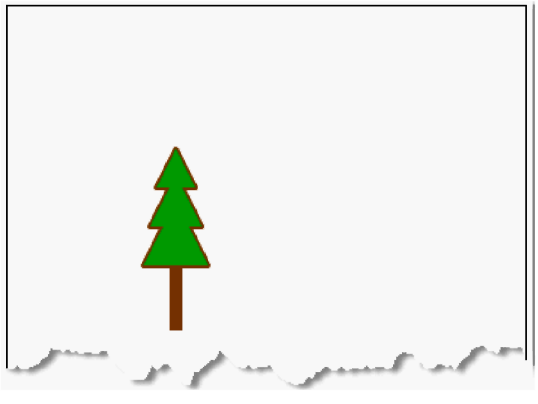


Рис. 2.9. Внешний вид дерева с закрашенным прямоугольным стволом

Рисование кривых

Образы внешнего мира, особенно окружающей природы, не ограничиваются одними только прямыми линиями и прямоугольниками. К счастью, элемент `canvas` предоставляет целый ряд функций, позволяющих создавать криволинейные участки путей. Ниже будет проиллюстрирована лишь простейший из доступных вариантов — *квадратичная* кривая, позволяющая создать тропинку в нашем виртуальном лесу. Листинг 2.13 демонстрирует добавление двух квадратичных кривых.

Листинг 2.13. Рисование кривых

```
// Сохранить состояние холста и создать путь
context.save();
context.translate(-10, 350);
context.beginPath();

// Первая кривая изгибается вверх и вправо
context.moveTo(0, 0);
context.quadraticCurveTo(170, -50, 260, -190);

// Вторая кривая продолжается вправо, плавно приближаясь
// к горизонтальной линии
context.quadraticCurveTo(310, -250, 410,-250);

// Прорисовать путь с использованием толстой коричневой линии
context.strokeStyle = '#663300';
context.lineWidth = 20;
context.stroke();

// Восстановить первоначальное состояние холста
context.restore();
```

Как и прежде, одной из первых операций, которые необходимо выполнить, является сохранение состояния контекста холста, так как параметры трансляции и обводки будут изменяться. Что касается самой лесной тропинки, то мы начинаем рисовать ее из начала координат и проводим первую квадратичную кривую в направлении вверх и вправо.

Как видно из рис. 2.10, функция `quadraticCurveTo()` начинает рисовать в текущей позиции, принимая в качестве параметров координаты `x` и `y` двух точек. Вторая точка — это конечная точка кривой, тогда как первая представляет *управляющую точку* (control point). Управляющая точка располагается в стороне от кривой (а не на ней) и действует подобно центру притяжения по отношению к точкам пути кривой. Изменяя положение управляющей точки, можно регулировать кривизну создаваемого пути. Вторая квадратичная кривая продолжает первую в направлении вверх и вправо, плавно приближаясь к горизонтальной линии, завершая путь. Далее остается только применить к пути обводку, как мы это делали с кроной дерева, с тем лишь отличием, что в данном случае используется большая толщина линии.

Другие возможные варианты построения кривых обеспечиваются в HTML5 Canvas API функциями `bezierCurveTo()`, `arcTo()` и `arc()`. Для определения характеристик кривых эти функции принимают в качестве аргументов координаты дополнительных точек, а также величины радиусов или углов. Внешний вид лесной тропики, нарисованной на холсте с использованием двух квадратичных кривых, показан на рис. 2.11.

Вставка изображений в элемент `canvas`

Вставка изображений в рисунок открывает чрезвычайно интересные возможности. При этом изображения могут просто помещаться на холст, растягиваться или изменяться с помощью преобразований и часто становятся центральным элементом всего рисунка. К счастью, в HTML5 Canvas API предусмотрено несколько простых функций, позволяющих добавлять изображения.

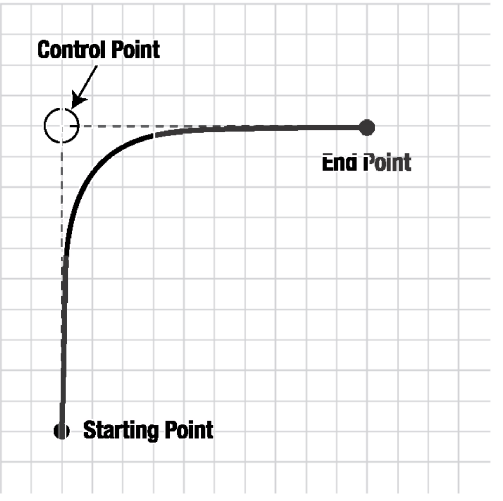


Рис. 2.10. Начальная, конечная и управляющая точки квадратичной кривой

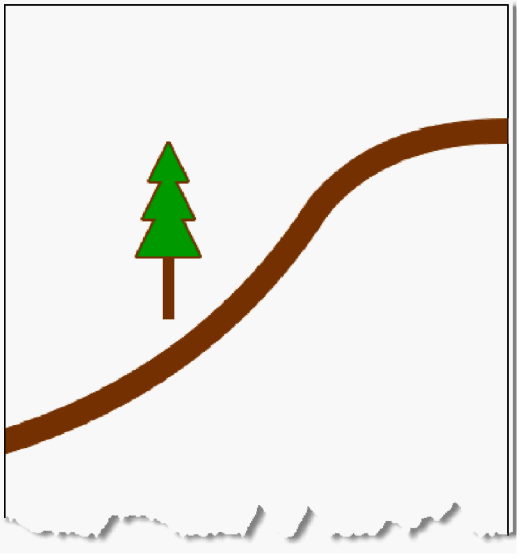


Рис. 2.11. Путь, построенный с использованием двух квадратичных кривых

Вместе с тем, добавление изображений несколько усложняет выполнение операций с холстом, поскольку изображение необходимо предварительно загрузить. Обычно браузеры загружают изображения асинхронно в процессе визуализации сценария страницы. Однако, если вы попытаетесь визуализировать изображение на холсте в тот момент, когда оно еще не будет полностью загружено, то холст не сможет визуализировать и любое другое изображение. Следовательно, необходимо

быть внимательным и тщательно следить за тем, чтобы визуализация изображений осуществлялась лишь после их полной загрузки.

В нашем примере таким изображением будет текстура коры для ствола дерева. Чтобы визуализация могла начаться только после того, как изображение полностью загрузится, поместим основной код в функцию обратного вызова, которая будет запускаться по завершении загрузки изображения, как показано в листинге 2.14.

Листинг 2.14. Загрузка изображения

```
// Загрузить изображение текстуры коры
var bark = new Image();
bark.src = "bark.jpg";

// Как только изображение загрузится, вывести рисунок на холст
bark.onload = function () {
  drawTrails();
}
```

Из этого листинга следует, что вызов основной функции `drawTrails()`, который мы поместили в обработчик событий `onload`, подключенный к изображению `bark.jpg`, произойдет лишь после полной загрузки изображения. Это гарантирует его доступность для последующего вызова функции, добавленного в код визуализации холста, как показано в листинге 2.15.

Листинг 2.15. Вывод изображения на холст

```
// Вывести изображение коры вместо закрашенного прямоугольника
context.drawImage(bark, -5, -50, 10, 50)
```

Здесь вместо прежнего вызова функции `fillRect()` используется вызов функции `drawImage()`, ответственной за вывод изображения коры, имитирующей ствол дерева. Хотя вносимые при этом изменения не сразу бросаются в глаза, дерево приобретает более естественный вид. Обратите внимание, что в качестве аргументов функции `drawImage()` помимо самого изображения указаны также его координаты `x` и `y`, ширина и высота. Выбранные значения параметров обеспечивают масштабирование изображения до размеров 10×50 пикселей, что соответствует размерам области, отведенной нами для ствола. Данной функции можно было бы дополнительно передать исходные размеры изображения, чтобы обеспечить отсечение только той его части, которая должна отображаться.

Как видно из рис. 2.12, использование изображения вместо закрашенного прямоугольника слегка изменило внешний вид ствола.

Использование градиентов

Вас не удовлетворяет, как выглядит ствол? Нас также. В таком случае давайте прибегнем к другому подходу и используем для прорисовки ствола более тонкий инструмент — градиенты. Градиенты обеспечивают постепенное изменение цветов в некотором направлении по определенному закону и могут применяться для стилевого оформления обводки и заливки, подобно шаблонным узорам, как это было сделано в предыдущем разделе. Для создания градиента необходимо выполнить следующие действия.

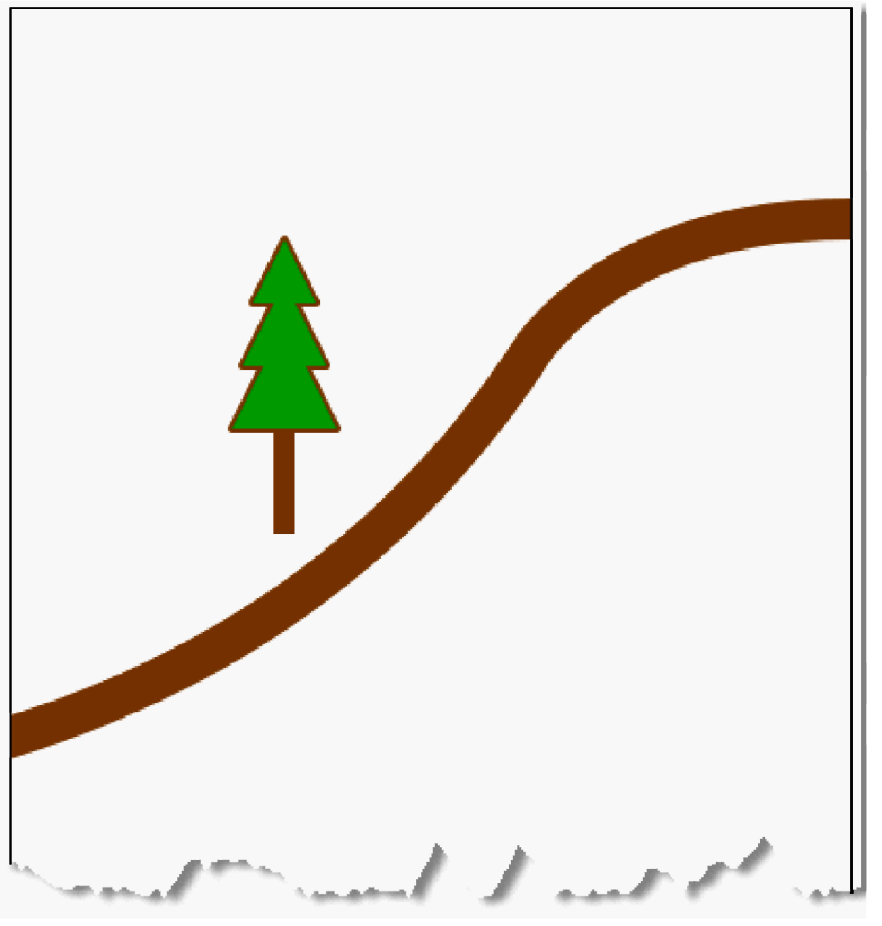


Рис. 2.12. Использование изображения для имитации ствола дерева

1. Создайте объект градиента.
2. Задайте цвета и смещения позиций остановки градиента в направлении цветового перехода.
3. Укажите градиент в качестве значений свойств `fillStyle` или `strokeStyle` данного контекста.

Возможно, проще всего рассматривать градиент как плавное изменение цвета при перемещении вдоль некоторой линии (оси градиента). Например, если при создании градиента в качестве аргументов задаются координаты точек `A` и `B`, то для любой линии или заливки будет создаваться цветовой переход вдоль оси градиента, направление которой совпадает с направлением от точки `A` к точке `B`.

Для определения отображаемых цветов используется метод `addColorStop()` объекта градиента. При вызове этого метода необходимо указать цвет и смещение позиции остановки градиента. Аргумент `color` устанавливает цвет линии или заливки в позиции, относительное смещение которой вдоль оси градиента определяется аргументом `offset`. Смещение может принимать значения в диапазоне от `0.0` до `1.0`. Чем ближе значение смещения к `0`, тем ближе к началу оси градиента рас-

полагается цвет. Цвета в промежуточных точках оси градиента определяются линейной интерполяцией цветов в соседних позициях остановки.

Например, если создать градиент в направлении от точки `0,0` к точке `0,100`, определив белый цвет для смещения `0.0` и черный — для смещения `1.0`, то при прорисовке линии или заливке области их цвет будет постепенно меняться с белого (начальная позиция остановки) на черный (конечная позиция остановки) по мере продвижения в направлении от точки `0,0` к точке `0,100`.

Как и в других случаях работы с цветом, при создании цветовых переходов можно дополнительно использовать альфа-канал, определяя наряду с параметрами цвета также значение альфа-компоненты (например, прозрачности). В этом случае цветовые параметры будут представляться с использованием другого текстового формата и задаваться, скажем, с помощью CSS-функции `rgba()`, одним из аргументов которой как раз и является альфа-канал.

Рассмотрим более подробно пример кода, в котором для представления окончательного варианта ствола дерева с помощью функции `fillRect()` используются два градиента, как показано в листинге 2.16.

Листинг 2.16. Использование градиента

```
// Создать 3 позиции остановки градиента в горизонтальном
// направлении (поперек ствола дерева)
var trunkGradient = context.createLinearGradient(-5, -50, 5, -50);

// Левый край ствола — умеренно-коричневый цвет
trunkGradient.addColorStop(0, '#663300');

// Ближе к середине ствола — более светлый цвет
trunkGradient addColorStop(0.4, '#996600');

// Правый край ствола самый темный
trunkGradient.addColorStop(1, '#552200');

// Применить градиент в стиле заливки и нарисовать ствол
context.fillStyle = trunkGradient;
context.fillRect(-5, -50, 10, 50);

// Второй (вертикальный) градиент создает на стволе тень от кроны
var canopyShadow = context.createLinearGradient(0, -50, 0, 0);

// Начало градиента тени -- черное, но с 50% значением
// альфа-канала
canopyShadow.addColorStop(0, 'rgba(0, 0, 0, 0.5)');

// Немного ниже градиент спадает до полной прозрачности.
// На остальную часть ствола тень не падает
canopyShadow.addColorStop(0.2, 'rgba(0, 0, 0, 0.0)');

// Нарисовать градиент тени поверх градиента ствола
context.fillStyle = canopyShadow;
context.fillRect(-5, -50, 10, 50);
```

Как показано на рис. 2.13, применение этих двух градиентов позволяет имитировать приятное мягкое освещение, падающее на дерево, благодаря чему создается эффект округлости ствола и отбрасывания тени кроной. Сохраним этот эффект.

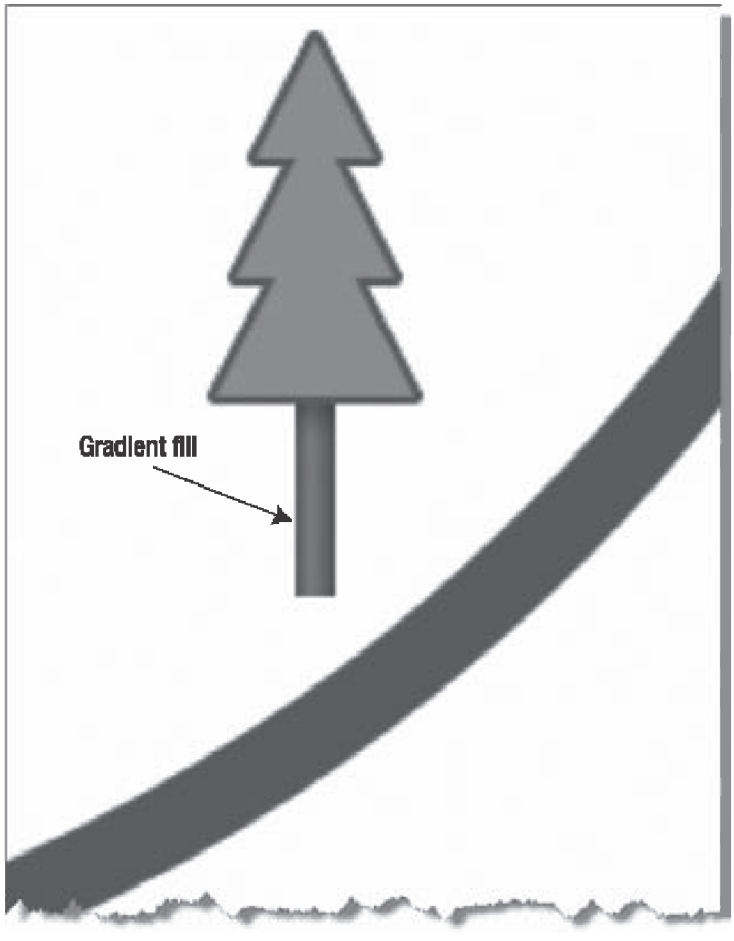


Рис. 2.13. Дерево с градиентной заливкой ствола

Спецификация HTML5 Canvas поддерживает не только линейные градиенты, продемонстрированные в нашем примере, но и радиальные, позволяющие задавать две окружности, в которых позиции остановки градиента применяются к построенному на указанных окружностях усеченному конусу. Радиальные градиенты используют те же позиции остановки, что и линейные, но принимают аргументы, представленные в листинге 2.17.

Листинг 2.17. Пример использования радиального градиента

```
createRadialGradient(x0, y0, r0, x1, y1, r1)
```

В этом вызове функции первые три аргумента представляют одну окружность с центром в точке с координатами `(x0,y0)` и радиусом `r0`, а последние три аргумента — вторую окружность с координатами `(x1,y1)` и радиусом `r1`. Градиент создается в области между окружностями.

Использование фоновых изображений

Непосредственная визуализация изображений имеет множество полезных применений, но в некоторых случаях изображение выгоднее использовать в качестве элемента мозаики, аналогично тому, как это позволяет делать CSS. Вы уже знаете, как установить сплошной цвет в качестве стиля обводки или заливки. В спецификации HTML5 Canvas предусмотрена возможность использования изображений в виде повторяющихся шаблонных узоров (орнаментов), которые могут применяться как при обводке путей, так и при заливке областей.

Воспользуемся указанной возможностью для того, чтобы лишить нашу лесную тропинку ее неестественной гладкости, применив в качестве обводки пути тропинки не кривую линию, как до этого, а мозаичную заливку фоновым изображением. С этой целью вместо изображения коры используем изображение гравия. Соответственно, вместо функции `drawImage()` будет вызываться функция `createPattern()`, как показано в листинге 2.18.

Листинг 2.18. Использование мозаичного фонового изображения

```
// Заменить изображение коры изображением гравия
var gravel = new Image();
gravel.src = "gravel.jpg";
gravel.onload = function () {
    drawTrails();
}

// Заменить сплошную линию повторяющимся фоновым изображением
context.strokeStyle = context.createPattern(gravel, 'repeat');
context.lineWidth = 20;
context.stroke();
```

Как нетрудно заметить, для создания пути по-прежнему вызывается функция `stroke()`. Однако на этот раз предварительно устанавливается свойство `strokeStyle` контекста, значением которого становится результат, возвращаемый функцией `createPattern()`. И конечно же, для выполнения операции необходимо предварительно загрузить изображение. Второй аргумент задает способ повторения шаблонного узора и может принимать одно из значений, перечисленных в табл. 2.2.

Таблица 2.2. Повторяющиеся шаблоны

Значение параметра `Repeat`	Описание
`repeat`	Повторение изображения в обоих направлениях (по умолчанию)
`repeat-x`	Повторение изображения в направлении оси X
`repeat-y`	Повторение изображения в направлении оси Y
`no-repeat`	Изображение не повторяется и не выводится

Результат использования готового фонового мозаичного шаблона вместо закрашивания тропинки представлен на рис. 2.14.

Масштабирование объектов холста

Ну, что это за лес, если в нем растет только одно дерево? Чтобы немного упростить себе работу, преобразуем код нашего примера, выделив операции, с помощью которых рисуется дерево, в отдельную функцию с именем `drawTree`, как показано в листинге 2.19.

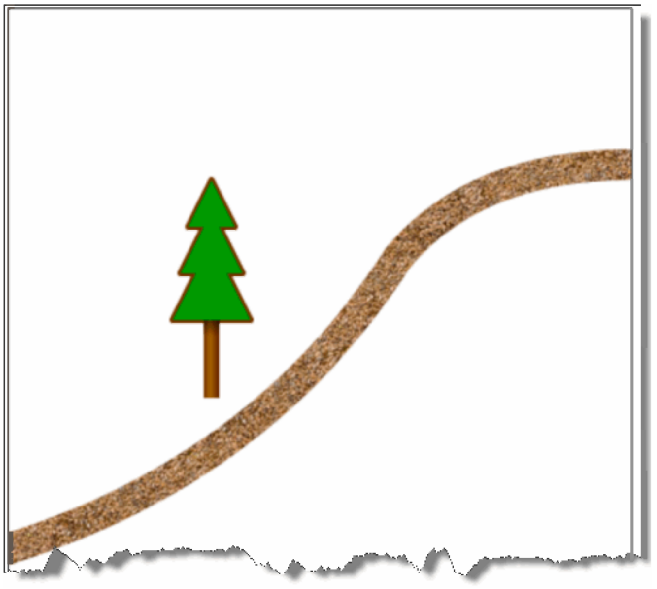


Рис. 2.14. Визуализация тропинки с помощью повторяющегося фонового изображения

Листинг 2.19. Функция для рисования объекта дерева

```
// Поместить код, ответственный за рисование дерева,
// в отдельную функцию, позволяющую многократно
// использовать этот код
function drawTree(context) {
  var trunkGradient = context.createLinearGradient(-5, -50, 5, -50);
  trunkGradient.addColorStop(0, '#663300');
  trunkGradient.addColorStop(0.4, '#996600');
  trunkGradient.addColorStop(1, '#552200');
  context.fillStyle = trunkGradient;
  context.fillRect(-5, -50, 10, 50);

  var canopyShadow = context.createLinearGradient(0, -50, 0, 0);
  canopyShadow.addColorStop(0, 'rgba(0, 0, 0, 0.5)');
  canopyShadow.addColorStop(0.2, 'rgba(0, 0, 0, 0.0)');
  context.fillStyle = canopyShadow;
  context.fillRect(-5, -50, 10, 50);

  createCanopyPath(context);

  context.lineWidth = 4;
  context.lineJoin = 'round';
  context.strokeStyle = '#663300';
  context.stroke();

  context.fillStyle = '#339900';
  context.fill();
}
```

В функцию `drawTree()` включен весь код, созданный нами ранее для рисования кроны, ствола и градиента ствола. Далее мы воспользуемся одной из функций преобразования — `context.scale()` — для прорисовки дерева в другом месте страницы, увеличив при этом его размеры, как показано в листинге 2.20.

Листинг 2.20. Рисование объектов деревьев

```
// Нарисовать первое дерево в точке с координатами X=130, Y=250
context.save();
context.translate(130, 250);
drawTree(context);
context.restore();

// Нарисовать второе дерево в точке с координатами X=260, Y=500
context.save();
context.translate(260, 500);

// Увеличить размеры второго дерева в два раза по сравнению
// с первым в обоих направлениях
context.scale(2, 2);
drawTree(context);
context.restore();
```

Двумя аргументами функции `scale()` служат масштабные коэффициенты вдоль осей `X` и `Y`. Эти коэффициенты указывают данному экземпляру холста, во сколько раз следует увеличить (уменьшить) размер в соответствующем направлении. Например, значение коэффициента для оси `X`, равное `2`, указывает на то, что во всех последующих графических операциях ширина объектов будет увеличиваться в два раза, тогда как значение коэффициента для оси `Y`, равное `0.5`, означало бы уменьшение высоты объектов в два раза. Использование этой функции позволило без особого труда создать на холсте второе дерево, как показано на рис. 2.15.

Всегда выполняйте подпрограммы для работы с фигурами и путями с привязкой к началу координат

Говорит (и на этот раз действительно серьезно) Брайан: "Этот пример хорошо объясняет одну из причин, по которым целесообразно сначала создавать пути и фигуры вблизи начала координат (начала отсчета), и только потом применять к ним нужные преобразования, как это было сделано в нашем коде с применением операции трансляции. Причина, по которой следует поступать именно так, заключается в том, что такие преобразования, как `scale` или `rotate`, выполняются относительно начала отсчета.

Если применить преобразование `rotate` к фигуре, которая не располагается компактно вблизи начала отсчета, а смещена от него, то в результате вы получите не поворот фигуры на месте, а поворот ее как целого вокруг начала отсчета. Точно так же, если применить преобразование `scale` к фигуре, удаленной от начала отсчета, то масштабированию подвергнутся не только линейные размеры фигуры, но и *координаты* всех ее точек относительно начала отсчета. В зависимости от величины масштабного коэффициента, новые местоположения точек пути могут вообще выйти за пределы холста, заставляя вас недоумевать по поводу того, что операция масштабирования "уничтожила" ваш объект."

Использование преобразований

Операции преобразования не ограничиваются масштабированием и трансляцией. Возможны также поворот графического контекста с помощью функции `context.rotate(`*угол*`)` и даже видоизменение самого преобразования для выполнения более сложных операций, например сдвига визуализированных путей. Если,

скажем, требуется повернуть изображение на некоторый угол, то для этого достаточно выполнить последовательность операций, представленную в листинге 2.21.

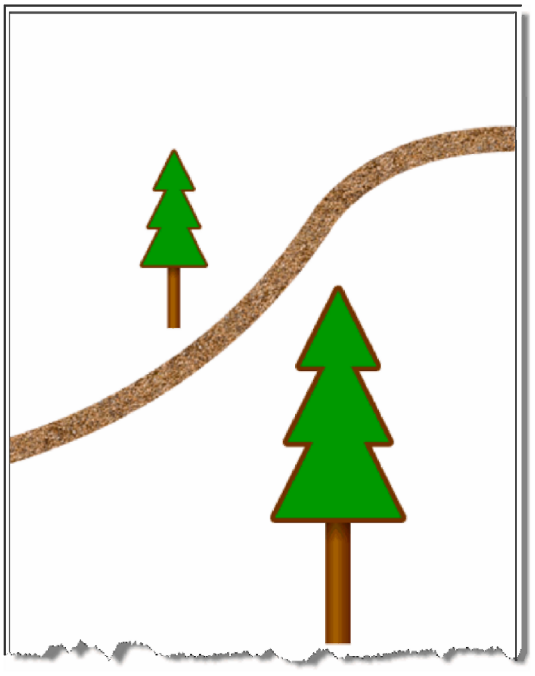


Рис. 2.15. Второе дерево с увеличенными в два раза размерами

Листинг 2.21. Поворот изображения

```
context.save();

// Угол поворота задается в радианах
context.rotate(1.57);
context.drawImage(myImage, 0, 0, 100, 100);

context.restore();
```

Более широкие возможности иллюстрирует листинг 2.22, в котором продемонстрировано, как применение произвольного преобразования к координатам пути способно настолько радикально изменить вид существующего изображения, что даже позволяет создать эффект тени.

Листинг 2.22. Использование преобразования

```
// Создать градиент с тремя позициями остановки
// в горизонтальном направлении (поперек дерева).
// Сохранить текущее состояние контекста для
// последующего использования.
context.save();

// Создать наклоненное дерево, имитирующее тень, путем
// применения преобразования сдвига, увеличивающего значения X
// по мере увеличения значений Y
```

```
context.transform(1, 0,-0.5, 1, 0, 0);

// Уменьшить размер тени в направлении Y до 60% от исходной высоты
context.scale(1, 0.6);

// Установить для дерева черную заливку, но со значением
// альфа-канала, равным всего лишь 20%
context.fillStyle = 'rgba(0, 0, 0, 0.2)';
context.fillRect(-5, -50, 10, 50);

// Перерисовать дерево, но уже с примененным эффектом тени
createCanopyPath(context);
context.fill();

// Восстановить состояние контекста
context.restore();
```

Пытаться применять к контексту преобразование в общем виде с помощью функции `context.transform()`, как это было сделано в приведенном примере, следует только в том случае, если вы хорошо понимаете математическую подоплеку, лежащую в основе 2D-графики. Если проанализировать математические формулы, описывающие данное преобразование, то можно увидеть, что в процессе применения преобразования сдвига к серому дереву, имитирующему тень, величины сдвига координат `x` его точек пропорциональны значениям их координат `y`. Последующее применение масштабного коэффициента 60% уменьшает высоту этого дерева.

Обратите внимание, что первой визуализируется "тень дерева", полученная в результате применения операции сдвига, так что собственно дерево оказывается расположенным выше тени в смысле *Z-порядка*, т.е. порядка наложения объектов. Кроме того, при рисовании тени дерева для указания его цвета используется CSS-нотация RGBA, что дало возможность использовать альфа-канал, для которого было установлено значение, составляющее лишь 20% от нормальной величины. Благодаря этому нам удалось изобразить тень в мягких полупрозрачных тонах. Полученный результат проиллюстрирован на рис. 2.16.

Использование текстовых функций объекта холста

Поскольку наша работа над созданием рисунка близится к завершению, продемонстрируем возможности текстовых функций HTML5 Canvas API, добавив в верхней части рисунка красивое титульное название. Важно отметить, что текст, визуализируемый на холсте, обрабатывается точно так же, как любой другой путь: к нему могут применяться обводка и заливка, а все стили и преобразования, используемые при визуализации, могут применяться к тексту точно так же, как к любой другой фигуре.

Как и можно было ожидать, возможности вывода текста представлены двумя методами объекта контекста:

- `fillText(`*`текст, x, y, макс_ширина`*`);`
- `strokeText(`*`текст, x, y, макс_ширина`*`).`

В качестве аргументов обе функции получают сам текст и координаты его местоположения. Также может быть предоставлен дополнительный аргумент *макс_ширина*, позволяющий ограничить ширину текста путем автоматического уменьшения размера шрифта до заданной величины. Кроме того, имеется допол-

нительная функция `measureText()`, которая возвращает метрический объект, содержащий ширину указанного текста в случае его визуализации с использованием текущих настроек контекста.

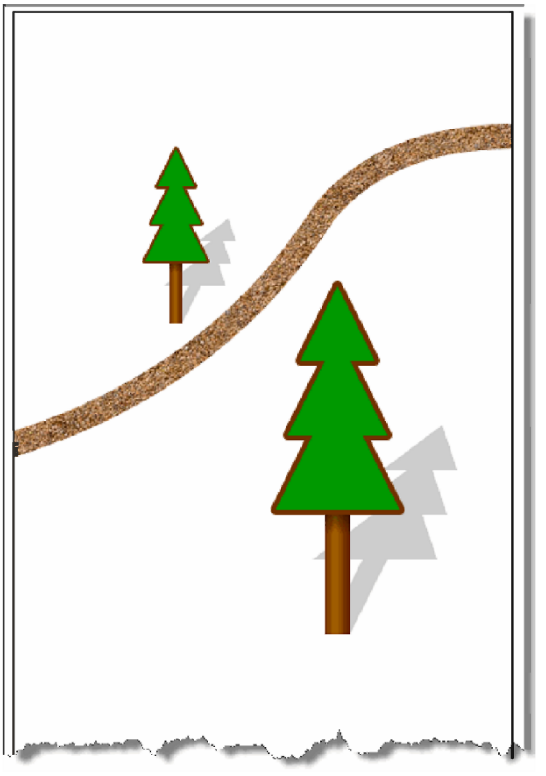


Рис. 2.16. Деревья с тенями, полученными методом преобразований

Как и во всех других случаях отображения текста в браузерах, характеристики его фактического внешнего вида можно изменять в широких пределах с помощью свойств контекста, сходных с их CSS-аналогами, как показано в табл. 2.3.

Таблица 2.3. Свойства объекта `context`, влияющие на внешний вид текста, отображаемого на холсте

Свойство	Значения	Примечания
`font`	Строка шрифта CSS	Пример: `italic Areal, sans-serif`
`textAlign`	`start`, `end`, `left`, `right`, `center`	По умолчанию `start`
`textBaseLine`	`top`, `hanging`, `middle`, `alphabetic`, `ideographic`, `bottom`	По умолчанию `alphabetic`

Вы можете либо устанавливать все эти свойства для изменения поведения контекста, либо обращаться к ним для получения текущих значений. В листинге 2.23 мы создаем текст с помощью крупного шрифта семейства `Impact` и закрашиваем буквы коричневым цветом. Центрирование текста в верхней части холста обеспечивается установкой подходящего значения *макс_ширина* и использованием выравнивания `center`.

Листинг 2.23. Использование текстовых функций холста

```
// Рисование титульного названия на холсте
context.save();

// Выбрать шрифт семейства Impact размером 60 пикселей
context.font = "60px impact";

// Использовать для текста коричневую заливку
context.fillStyle = '#996600';
// Возможна установка выравнивания текста при его отображении
context.textAlign = 'center';

// Нарисовать текст посередине холста с подходящим значением
// параметра "макс_ширина" для обеспечения центрирования
context.fillText('Happy Trails!', 200, 60, 400);
context.restore();
```

Как вы сами можете судить по результату, представленному на рис. 2.17, добавление текста действительно оживило рисунок.

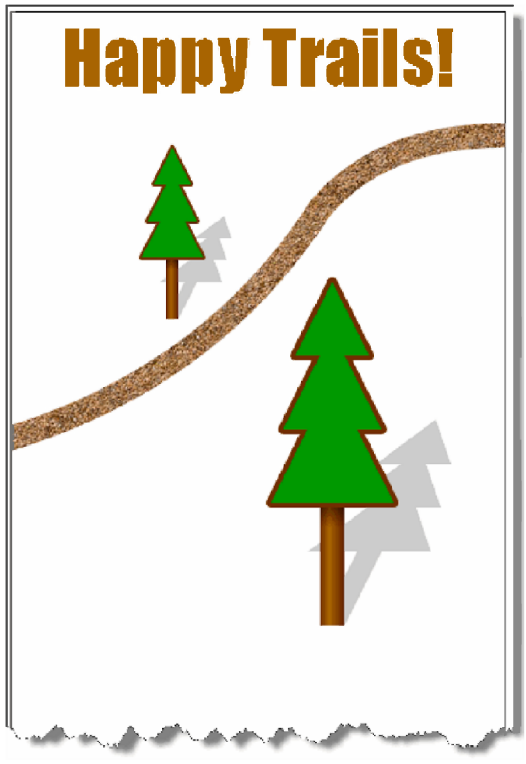


Рис. 2.17. Текст, закрашенный сплошным цветом

Применение теней

Наконец, мы добавим к тексту размытую тень, воспользовавшись встроенными возможностями холста. Несмотря на то что спецификация HTML5 Canvas позволяет применять тени вместе с любой из уже рассмотренных до этого операций, в от-

ношении использования теней, как и многих других эффектов, необходимо проявлять осторожность.

Управление тенями осуществляется с помощью нескольких свойств контекста, представленных в табл. 2.4.

Таблица 2.4. Свойства, определяющие характеристики тени

Свойство	Значение	Примечания
`shadowColor`	Любое CSS-значение цвета	Допускается включение альфа-канала
`shadowOffsetX`	Смещение (в пикселях) по оси X	Положительным значениям соответствует смещение тени вправо, отрицательным — влево
`shadowOffsetY`	Смещение в (пикселях) по оси Y	Положительным значениям соответствует смещение тени вниз, отрицательным — вверх
`shadowBlur`	Размытие по Гауссу	Большим значениям соответствуют более размытые края тени

Эффект тени срабатывает при визуализации любого пути, текста или изображения, если для свойства `shadowColor` и по крайней мере одного из других указанных в таблице свойств установлены значения, отличные от заданных по умолчанию. Листинг 2.24 демонстрирует применение тени к тексту названия нашего рисунка.

Листинг 2.24. Применение тени к тексту

```
// Задать для текста тень черного цвета со значением
// альфа-канала, равным 20%
context.shadowColor = 'rgba(0, 0, 0, 0.2)';

// Переместить тень вправо на 15 пикселей и вверх на 10 пикселей
context.shadowOffsetX = 15;
context.shadowOffsetY = -10;

// Слегка размыть тень
context.shadowBlur = 2;
```

После внесения этих несложных изменений механизм визуализации холста будет автоматически применять тени до тех пор, пока не будет восстановлено прежнее состояние холста или не будут сброшены значения свойств, определяющих характеристики тени. Результат применения новых теней представлен на рис. 2.18.

Как видно из рисунка, углы падения этой тени и тени, сгенерированной с помощью преобразований, оказались разными. Чтобы обеспечить согласование всех элементов рисунка между собой, будет, пожалуй, лучше, если в процессе создания теней для любой графической сцены вы будете использовать только один определенный подход.

Работа с пиксельными данными

Одним из наиболее полезных аспектов программного интерфейса HTML5 Canvas является то, что с его помощью разработчики могут легко получать доступ к пикселям, образующим холст. Предоставляемый доступ является двусторонним в том смысле, что насколько тривиально получить доступ к значениям пикселей в виде числового массива, настолько же просто можно изменить эти значения и применить их к холсту. Фактически, вполне возможно манипулировать всем холстом исключительно посредством функций для работы с пикселями, полностью отказавшись от функций для построения графических объектов, обсуждавшихся

в этой главе. Такая возможность связана с существованием у объекта `context` трех функций.

Рис. 2.18. Титульное название, отбрасывающее тень

Первая из них — функция `context.getImageData(sx,sy,sw,sh)`. Она возвращает текущее состояние изображения на холсте в виде коллекции целых чисел. В частности, она возвращает объект, содержащий следующие три свойства.

- `width`. Количество пикселей в каждой строке пиксельных данных.
- `height`. Количество пикселей в каждом столбце пиксельных данных.
- `data`. Одномерный массив, содержащий фактические RGBA-значения для каждого пикселя, извлекаемого из холста. Этот массив содержит по четыре значения для каждого пикселя, соответствующие красному, зеленому, синему цветам и альфа-каналу, значения которых лежат в диапазоне от 0 до 255. Поэтому каждый пиксель, извлеченный из холста, представляется в указанном массиве данных четырьмя целыми числами. Заполнение массива пикселями происходит в направлении слева направо и сверху вниз (например, сначала заполняется первая строка, затем — вторая и т.д.), как показано на рис. 2.19.

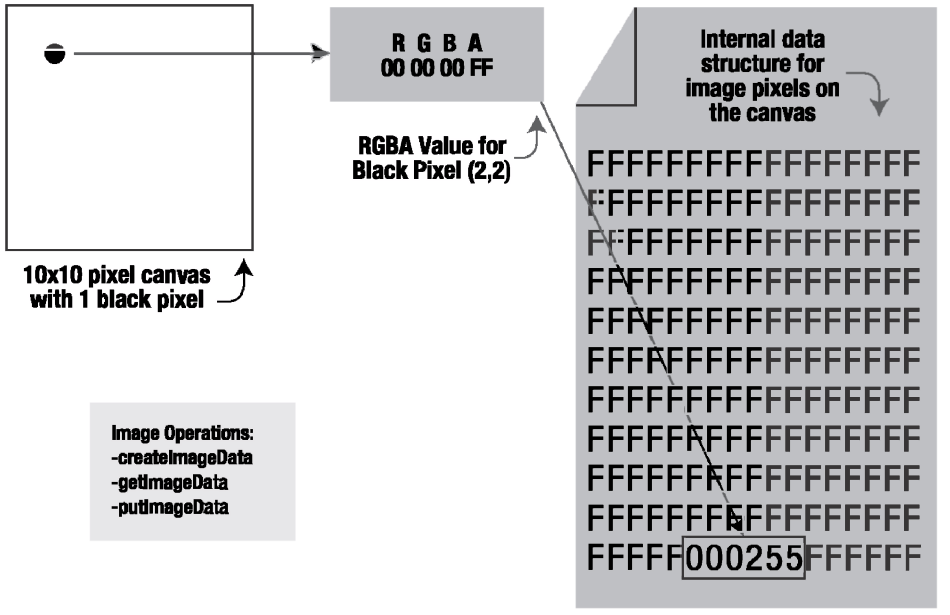


Рис. 2.19. Пиксельные данные и используемая для их представления внутренняя структура данных

Данные, возвращаемые функцией `getImageData()`, ограничиваются областью, определяемой четырьмя параметрами. Поэтому извлекаться будут лишь пиксели холста, содержащиеся внутри прямоугольника, определенного значениями параметров `x`, `y`, `width` и `height`. Следовательно, чтобы извлечь в качестве данных все пиксели, необходимо использовать такой вызов:

```
getImageData(0, 0, canvas.width, canvas.height)
```

Поскольку каждый пиксель представлен в массиве данных четырьмя числами, точный расчет индекса элемента массива, соответствующего определенной компоненте данного пикселя, требует тщательности. Рассмотрим используемые для этого формулы.

Для пикселя с координатами `(x,y)` на холсте с заданными шириной (width) и высотой (height) индексы значений каждой из цветовых компонент в одномерном массиве данных определяются по следующим формулам.

- **Красная компонента:** ((width * y) + x) * 4.
- **Зеленая компонента:** ((width * y) + x) * 4 + 1.
- **Синяя компонента:** ((width * y) + x) * 4 + 2.
- **Альфа-компонента:** ((width * y) + x) * 4 + 3.

Как только получен доступ к объекту, содержащему данные изображения, изменение пиксельных значений в массиве данных с математической точки зрения не представляет никакой сложности, поскольку все они являются целыми числами, имеющими значения в диапазоне от 0 до 255. Обновление изображения на холсте путем изменения значений, соответствующих красному, зеленому, синему цветам и альфа-каналу, для одного или нескольких пикселей осуществляется с помощью второй из упомянутых выше трех функций:

```
context.putImageData(imagedata, dx, dy)
```

Данные изображения передаются этой функции в том же формате, в котором они первоначально извлекались. Это довольно удобно, поскольку после изменения полученных от холста значений их можно просто вернуть обратно холсту. Вызов этой функции приводит к немедленному обновлению холста, изображение на котором теперь будет соответствовать новым пиксельным значениям, переданным в качестве данных изображения. Параметры dx и dy позволяют при необходимости указать смещение позиции, начиная с которой данные массива должны быть применены к существующему холсту.

Наконец, если вы хотите начать с совершенно чистого холста, можете использовать функцию `context.createImageData(sw, sh)` для создания нового набора данных изображения, связанных с объектом холста. Хотя этот набор данных, если его извлечь, и не отражает текущего состояния холста, его можно изменить программными средствами, как было показано выше.

Обеспечение безопасности холста

В отношении манипулирования пикселями, о чем говорилось в предыдущем разделе, необходимо сделать одно важное замечание. Хотя подобные манипуляции в большинстве случаев используются как вполне законное средство разработки, нельзя исключить того, что возможность извлечения данных из холста или внесения в него изменений может быть использована в злонамеренных целях. По этой причине было введено понятие *чистоты происхождения* (origin-clean) холста, а извлечение данных из холстов, *загрязненных* (tainted) изображениями, которые поступили от источников, отличных от источника самой страницы, сделано невозможным.

Как показано на рис. 2.20, если страница, обслуживаемая сайтом `www.example.com`, содержит элемент `canvas`, то не исключено, что код страницы может пытаться визуализировать на холсте изображение с сайта `www.remote.com`. В конце концов, в попытках визуализировать на веб-странице изображения, полученные с удаленных сайтов, нет ничего предосудительного.

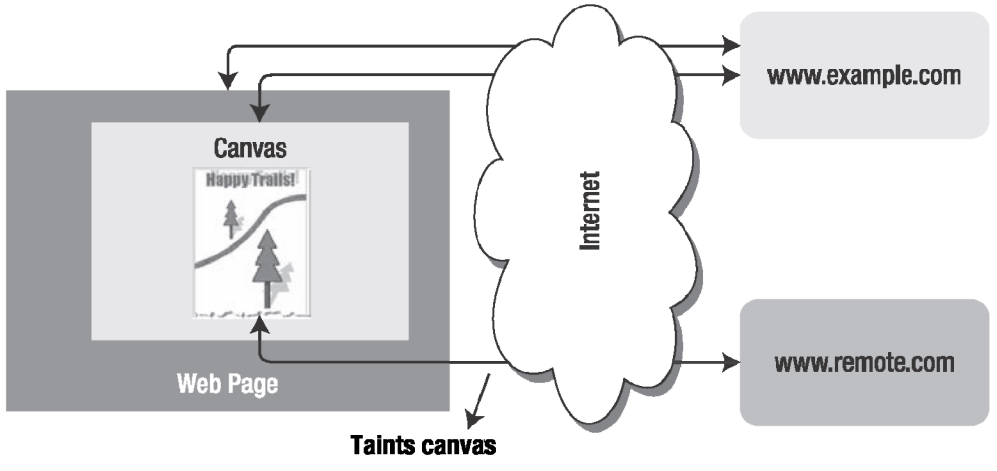


Рис. 2.20. Локальные и удаленные источники изображений

В то же время, до появления спецификации HTML5 Canvas извлечение пиксельных значений загруженного изображения программными средствами было невозможно. Коммерческие изображения, поступающие с других сайтов, можно было отображать на странице, но не считывать или копировать. Предоставление сцена-

риям возможности считывать данные изображений из других источников фактически означало бы открытие доступа к личным фотографиям пользователя и другим критически важным файлам изображений для всех пользователей Интернета.

Чтобы этого избежать, любой элемент `canvas`, который содержит изображения, визуализированные из удаленных источников, в случае вызова функции `getImageData()` будет генерировать исключение безопасности. Вполне допустимо визуализировать на холсте изображения из другого источника, коль скоро вы (или какой-то другой разработчик сценариев) не пытаетесь извлечь данные из холста, после того как он был "загрязнен". Не забывайте об этом ограничении и придерживайтесь практики безопасной визуализации.

Создание приложения с использованием программного интерфейса HTML5 Canvas

Существует множество областей применения спецификации HTML5 Canvas: построение графиков и диаграмм, редактирование изображений и т.д. Однако одним из самых интригующих применений элемента `canvas` является изменение или наложение существующих изображений. К популярным типам наложения изображений относятся так называемые *тепловые карты* (heatmap). Хотя это название наводит на мысль о температурных измерениях, в данном случае под ним можно подразумевать измерение любого вида активности. Области карты, соответствующие высоким уровням активности, окрашиваются "горячими" цветами (например, красным, желтым или белым). Области меньшей активности характеризуются полным отсутствие цветовых изменений или оттенками черного и серого цветов минимальной интенсивности.

Так, тепловую карту можно использовать для индикации интенсивности дорожного движения на городской карте или штормовой активности на карте мира. Ситуации, подобные этим, легко реализовать в HTML5, комбинируя изображение на холсте с источником данных для карты. По сути, холст можно использовать для наложения на него карты и рисования изотемпературных точек, исходя из соответствующих данных, описывающих активность.

Построим простую тепловую карту, используя возможности программного интерфейса Canvas, о которых вы только что узнали. В данном случае источником данных для нас будет служить не какой-либо внешний источник, а перемещение указателя мыши по карте. Прохождение указателя над любым участком карты будет вызывать его "нагрев", а если задержать указатель над каким-либо участком, то "температура" этого участка быстро повысится до максимального уровня. Для примера нам будет достаточно наложить соответствующее тепловое изображение на символическую карту местности (рис. 2.21).

Теперь, когда вы увидели результат работы приложения, перейдем к рассмотрению соответствующего кода. Как обычно, коды примеров, доступные для загрузки и изучения, находятся на сайте книги.

Начнем с HTML-элементов, объявленных в этом примере. В данном случае в их число входят титульное название, элемент `canvas` и кнопка, которую можно использовать для сброса тепловой карты. В качестве фона холста используется простое изображение `mapbg.jpg`, применяемое к холсту посредством стилей CSS, как показано в листинге 2.25.

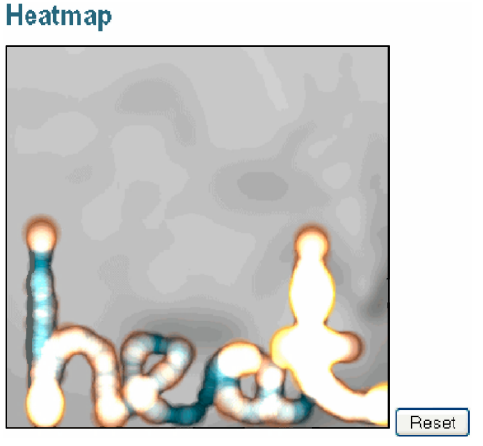


Рис. 2.21. Тепловая карта

Листинг 2.25. Элемент `canvas` тепловой карты

```
<style type="text/css">
  #heatmap {
      background-image: url("mapbg.jpg");
  }
</style>

<h2>Heatmap </h2>
<canvas id="heatmap" class="clear" style="border: 1px solid ; "
  height="300" width="300"> </canvas>
<button id="resetButton">Reset</button>
```

Мы также инициализируем некоторые переменные, которые потребуются нам позже:

```
var points = {};
var SCALE = 3;
var x = -1;
var y = -1;
```

Далее мы задаем высокое значение прозрачности при выполнении на холсте графических операций и устанавливаем композитный режим, чтобы новые рисунки ослабляли интенсивность пикселей, на которые они налагаются, а не замещали их.

После этого, как видно из листинга 2.26, мы устанавливаем обработчик событий `addToPoint()`, изменяющий изображение на холсте при каждом перемещении мыши или образовании паузы длительностью в одну десятую долю секунды.

Листинг 2.26. Функция `loadDemo()`

```
function loadDemo() {
  document.getElementById("resetButton").onclick = reset;

  canvas = document.getElementById("heatmap");
  context = canvas.getContext('2d');
  context.globalAlpha = 0.2;
  context.globalCompositeOperation = "lighter"
```

```
function sample() {
  if (x != -1) {
    addToPoint(x,y)
  }
  setTimeout(sample, 100);
}

canvas.onmousemove = function(e) {
  x = e.clientX - e.target.offsetLeft;
  y = e.clientY - e.target.offsetTop;
  addToPoint(x,y)
}

  sample();
}
```

При щелчке на кнопке Reset (Сброс) вся поверхность холста очищается и восстанавливает свое исходное состояние в результате вызова функции `clearRect()`, как показано в листинге 2.27.

Листинг 2.27. Функция `reset()`

```
function reset() {
  points = {};
  context.clearRect(0,0,300,300);
  x = -1;
  y = -1;
}
```

Далее мы создаем поисковую таблицу цветов, используемую для отображения тепловых узоров на холсте. Из листинга 2.28 видно, что яркость цветов может меняться в пределах определенного диапазона, и эти цвета будут использоваться для представления различных уровней "температуры" в виде изображений на холсте. Чем больше значение параметра `intensity`, тем ярче выглядит возвращаемый цвет.

Листинг 2.28. Функция `getColor()`

```
function getColor(intensity) {
  var colors = ["#072933", "#2E4045", "#8C593B", "#B2814E",
   "#FAC268", "#FAD237"];
  return colors[Math.floor(intensity/2)];
}
```

Всякий раз, когда указатель мыши перемещается над какой-либо областью холста или находится в ней, рисуется точка. Размеры (и яркость) точки увеличиваются в зависимости от длительности пребывания указателя мыши в непосредственной близости от этого места. Как показано в листинге 2.29, этот эффект создается в результате рисования окружностей с помощью функции `context.arc()`, причем чем больше величина радиуса, тем более яркий, "горячий" цвет, используемый для рисования указанных окружностей, возвращает функция `getColor()`.

Листинг 2.29. Функция `drawPoint()`

```
function drawPoint(x, y, radius) {
  context.fillStyle = getColor(radius);
  radius = Math.sqrt(radius)*6;
  context.beginPath();
```

```
  context.arc(x, y, radius, 0, Math.PI*2, true)
  context.closePath();
  context.fill();
}
```

В функции `addToPoint()`, обращение к которой, как вы помните, происходит всякий раз, когда указатель мыши перемещается или задерживается над какой-либо точкой, температурное значение увеличивается и сохраняется для данной точки холста. Как видно из листинга 2.30, максимальное значение температуры составляет `10`. Как только получено текущее значение температуры для данного пикселя, этот пиксель и соответствующее ему значение температуры (радиуса) передаются функции `drawPoint()`.

Листинг 2.30. Функция `addToPoint()`

```
function addToPoint(x, y) {
  x = Math.floor(x/SCALE);
  y = Math.floor(y/SCALE);
  if (!points[[x,y]]) {
    points[[x,y]] = 1;
  } else if (points[[x,y]]==10) {
    return
  } else {
    points[[x,y]]++;
  }
  drawPoint(x*SCALE,y*SCALE, points[[x,y]]);
}
```

Наконец, мы регистрируем функцию `loadDemo()` в качестве обработчика событий, который будет вызываться всякий раз при завершении загрузки окна.

```
window.addEventListener("load", loadDemo, true);
```

Описанный пример, весь код которого умещается в какой-то сотне строк, наглядно демонстрирует, сколь многого можно добиться с помощью программного интерфейса HTML5 Canvas в кратчайшие строки без привлечения каких-либо подключаемых модулей или технологий визуализации сторонних разработчиков. Теперь визуализация любого из бесчисленного множества доступных источников данных не составит для вас никакого труда.

Дополнительные рекомендации: полностраничная прозрачная панель

Рассмотренный пример приложения продемонстрировал один из возможных способов применения холста, налагаемого поверх графики. Однако возможно также применение холста поверх всего окна браузера или только какой-то его части. Соответствующая методика носит название *прозрачной*, или *стеклянной*, *панели* (glass pane). Размещение холста прозрачной панели поверх веб-страницы открывает перед разработчиком множество необычных, но чрезвычайно полезных возможностей.

Например, можно воспользоваться функцией, которая получает абсолютные позиции всех DOM-элементов на странице, и создать интерактивную программу, которая последовательно проведет пользователя веб-приложения через все этапы его запуска и применения.

Или же можно использовать холст прозрачной панели для того, чтобы надписывать вручную ссылки на веб-страницы других сайтов путем обработки событий мыши для рисования ввода. Пытаясь применить холст подобным образом, не упускайте из виду следующие моменты.

- Необходимо установить для холста абсолютное позиционирование и задать конкретные значения его позиции, ширины и высоты. Без явного определения ширины и высоты холст будет иметь нулевые размеры.
- Не забудьте установить для холста высокое значение Z-индекса, чтобы он всегда располагался над всем видимым содержимым. Визуализация холста, заслоненного существующим содержимым, не позволит ему раскрыться в полной мере.
- Холст прозрачной панели может блокировать доступ к событиям находящегося под ним содержимого, поэтому используйте его экономно и удаляйте, когда необходимость в нем исчезает.

Резюме

Как вы могли убедиться, программный интерфейс HTML5 Canvas предоставляет весьма широкие возможности для изменения внешнего вида веб-приложений без привлечения каких-либо нестандартных методов модифицирования документа. Совместное использование изображений, градиентов и сложных путей позволяет представить содержимое практически в любом виде. Не забывайте о том, что фигуры, как правило, необходимо рисовать с привязкой к началу координат; загружайте изображения, которые хотите отобразить, до того, как будет предприниматься попытка их визуализации; будьте осмотрительны при размещении на своем холсте изображений, полученных от посторонних источников. Вместе с тем, если вы овладеете навыками работы с холстом, то сможете создавать приложения, реализация которых на веб-страницах ранее была невозможна.

Глава 3

Работа со звуком и видео в HTML5

В этой главе мы изучим свойства двух важных элементов HTML5 — `audio` и `video` — и продемонстрируем перспективы их использования для создания профессиональных приложений. Обогащая HTML5 новыми возможностями для работы с мультимедиа, эти элементы позволяют использовать аудио- и видеосодержимое без привлечения подключаемых модулей и одновременно предоставляют обычный встроенный программный интерфейс, доступный из сценариев.

Сначала мы проанализируем текущее состояние дел с файловыми контейнерами и кодеками, используемыми для работы со звуком и видео. Затем мы коснемся вопроса об отсутствии общей поддержки кодеков — самом главном недостатке, затрудняющем работу с мультимедиа, — и объясним, почему можно надеяться на то, что в будущем эта проблема перестанет быть такой актуальной. Также будет рассказано о механизме, позволяющем автоматически переключаться на содержимое, тип которого лучше всего подходит для отображения в браузере.

Далее мы рассмотрим методы программного управления звуком и видео с помощью встроенных функций и продемонстрируем использование элементов `audio` и `video` на конкретных примерах.

Обзор возможностей элементов audio и video в HTML5

В следующих разделах мы рассмотрим некоторые ключевые понятия, без знакомства с которыми невозможно обсуждать использование элементов `audio` и `video` в HTML5: контейнеры и кодеки.

Видеоконтейнеры

На самом деле любой звуковой или видеофайл является всего лишь файловым *контейнером*, в котором, как в случае архивных ZIP-файлов, хранится целый ряд других файлов. На рис. 3.1 в схематическом виде представлена структура видеофайла (видеоконтейнера), содержащего аудиодорожки, видеодорожки и дополнительные метаданные. Аудио- и видеодорожки объединяются в среде выполнения для воспроизведения видеоролика. Метаданные содержат относящуюся к видеоролику информацию: изображение обложки, титры, субтитры и т.д.

Популярными форматами видеоконтейнеров являются следующие:

- Audio Video Interleave (.avi);
- Flash Video (.flv);
- MPEG 4 (.mp4);

- Matroska (.mkv);
- Ogg (.ogv).

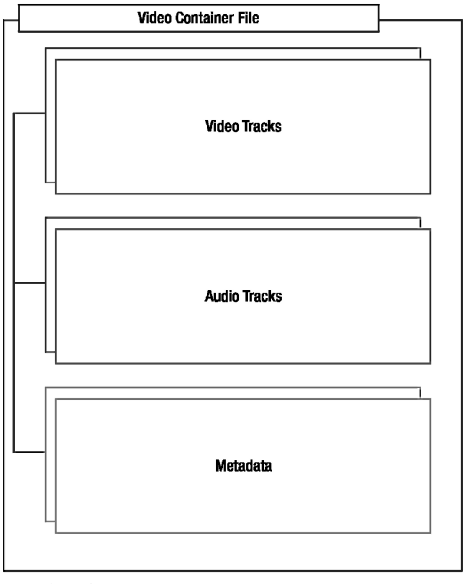


Рис. 3.1. Схематическая структура видеоконтейнера

Аудио- и видеокодеки

Кодеры/декодеры (*кодеки*) звука и видео — это алгоритмы, используемые для кодирования и декодирования определенных типов звуковых и видеопотоков с целью их воспроизведения. Файлы, содержащие необработанную мультимедийную информацию, имеют огромные размеры, поэтому без применения кодирования объемы данных аудио- и видеоклипов были бы настолько велики, что это затрудняло бы или вообще делало невозможной их передачу через Интернет за разумные промежутки времени. Каждый кодек "понимает" формат определенного контейнера и декодирует содержащиеся в нем аудио- и видеодорожки.

В качестве примера можно привести следующие аудиокодеки:

- AAC;
- MPEG-3;
- Ogg Vorbis.

А вот некоторые видеокодеки:

- H.264;
- VP8;
- Ogg Theora.

Война кодеков и временное перемирие

Одни кодеки защищены патентами, другие распространяются свободно. Например, аудиокодек Vorbis и видеокодек Theora — открытые, тогда как использование кодеков MPEG-4 и H.264 требует лицензионных отчислений.

Первоначально спецификация HTML5 предполагала обязательную поддержку некоторых кодеков. Однако часть производителей отказалась от включения поддержки кодека Ogg Theora в свои браузеры из-за низкого качества и отсутствия аппаратного ускорения для него. Например, в смартфоне iPhone корпорации Apple предусмотрено аппаратное ускорение декодирования для видео H.264, но не для Theora. С другой стороны, включение платных патентованных кодеков в свободные системы может затруднять распространение производных продуктов. Но в первую очередь именно низкие эксплуатационные характеристики некоторых проприетарных кодеков делают открытые кодеки более предпочтительными в глазах производителей браузеров. Таким образом, создалась тупиковая ситуация, когда невозможно выбрать какой-то один кодек, который согласились бы поддерживать все производители браузеров.

В настоящее время всякие упоминания об обязательной поддержке конкретных кодеков изъяты из спецификации, однако в будущем это решение может быть пересмотрено. А пока что вы должны ориентироваться в текущей поддержке кодеков браузерами и понимать, что, возможно, вам придется перекодировать свои мультимедийные данные для различных сред. (Вполне вероятно, что вы это уже делаете.)

Мы надеемся, что со временем поддержка различных кодеков будет расширяться и станет более унифицированной, что должно облегчить выбор общих типов мультимедиа, пригодных для повсеместного использования. Также не исключено, что преобладание одного из кодеков станет настолько очевидным, что он превратится в стандарт для Интернета. Кроме того, мультимедийные дескрипторы обладают встроенным механизмом, который обеспечивает переключение на тип содержимого, являющийся наиболее подходящим для отображения в браузере, что упростит поддержку различных сред.

Появление WebM

Говорит Фрэнк: "В мае 2010 года компания Google представила видеоформат WebM — новый открытый видеоформат, который должен прояснить туманную ситуацию с форматами, сложившуюся в Интернете. WebM-файлы имеют расширение `.webm`, а сам формат основан на видеокодеке VP8, аудиокодеке Ogg Vorbis и подмножестве медиаконтейнера Matroska. Google выпустила спецификацию и программное обеспечение WebM на условиях разрешительных лицензий, охватывающих исходный код и патентные права. Будучи форматом высокого качества, свободно доступным как для реализаторов, так и для издателей, видеоформат WebM ознаменовал значительный шаг вперед на рынке кодеков.

Что касается браузеров, то собственную поддержку WebM будут предоставлять по крайней мере Firefox, Opera и Chrome. Браузер Opera 10.60 уже поставляется с поддержкой WebM. Компании Mozilla и Google заявили о поставке WebM в следующих версиях своих браузеров."

Ограничения, действующие при использовании элементов `audio` и `video`

Ниже приводится перечень мультимедийных возможностей, поддержка которых элементами `audio` и `video` в спецификации HTML5 не предусмотрена.

- *Потоковые* звук и видео. В настоящее время отсутствует стандарт, регламентирующий переключение битрейта в элементе `video` в HTML5; текущие реализации поддерживают лишь полные мультимедийные файлы. Однако в спецификации имеются положения, предусматривающие поддержку потокового звука и видео в будущем, как только будет обеспечена поддержка соответствующих форматов.

- На использование мультимедийной информации распространяются ограничения, связанные с кроссдоменным разделением ресурсов (cross-origin resource sharing). Более подробно об этом говорится в главе 5.
- Полноэкранное видео не может запускаться из сценариев, поскольку предоставление элементу, управляемому сценарием, возможности захвата всего экрана недопустимо из соображений безопасности. Однако браузеры обеспечивают для пользователей возможность выбора полноэкранного режима просмотра видео путем предоставления дополнительных элементов управления.
- Обеспечение доступности элементов `audio` и `video` для людей с ограниченными возможностями еще не полностью специфицировано. В настоящее время ведется работа по созданию спецификации WebSRT, регламентирующей поддержку субтитров на основе популярного формата SRT.

Поддержка элементов audio и video браузерами

Как видно из табл. 3.1, на момент написания этой книги элементы `audio` и `video` в HTML5 уже поддерживались многими браузерами. В таблице также содержатся сведения о поддерживаемых кодеках.

Таблица 3.1. Поддержка элемента HTML5 `video` браузерами

Браузер	Наличие поддержки	Поддерживаемые кодеки и контейнеры
Chrome	Версия 3.0 и выше	Theora и Vorbis, контейнер Ogg, H.264 и AAC, контейнер MPEG
Firefox	Версия 3.5 и выше	Theora и Vorbis, контейнер Ogg
Internet Explorer	Поддержка отсутствует	Поддержка отсутствует
Opera	Версия 10.5 и выше	Theora и Vorbis, контейнер Ogg (версия 10.5 и выше)
Safari	Версия 3.2 и выше	H.264 и AAC, контейнер MPEG 4

Всегда имеет смысл предварительно проверять, поддерживает ли браузер дескрипторы HTML5 `audio` и `video`. О том, как это сделать программным путем, рассказано далее.

Программный интерфейс элементов audio и video

В этой главе исследуются возможности использования элементов `audio` и `video` в приложениях. По сравнению с известными методиками внедрения видео, связанными, как правило, с применением таких подключаемых модулей, как Flash, QuickTime или Windows Media, использование медиадескрипторов HTML5 обеспечивает следующие два преимущества, сулящие облегчение жизни как пользователям, так и разработчикам.

- *Новые дескрипторы `audio` и `video` устраняют препятствия, которые могут возникать при развертывании приложений, поскольку являются компонентами собственной среды браузера.* Несмотря на высокий рейтинг установки ряда подключаемых модулей, в управляемых корпоративных средах их часто блокируют. Некоторые пользователи предпочитают отключать их из-за отображаемой ими назойливой рекламы, тем самым лишая возможности использовать их для воспроизведения полезной мультимедийной информации. Кроме того, подключаемые модули могут создавать потенциальные бреши в системе безопасности. Наконец, нередко возникающие во мно-

гих случаях трудности с интеграцией вывода, генерируемого подключаемыми модулями и остальной частью содержимого, могут порождать на некоторых сайтах проблемы с отсечением или прозрачностью. Поскольку подключаемые модули используют независимую модель визуализации, отличную от модели визуализации базовых веб-страниц, разработчик столкнется с трудностями, если всплывающие меню или другие визуальные элементы пересекают границы плагина на странице.

- *Мультимедийные элементы предоставляют документу обычный встроенный программный интерфейс, управляемый из сценариев.* Используя новые мультимедийные элементы, разработчик может весьма просто организовать управление содержимым и его воспроизведением непосредственно из сценариев. Соответствующие многочисленные примеры приводятся далее.

Несомненно, использованию рассматриваемых мультимедийных элементов свойствен один существенный недостаток — отсутствие общей поддержки кодеков, о чем говорилось в предыдущих разделах. Однако можно надеяться, что с течением времени поддержка кодеков будет только улучшаться и расширяться, что упростит выбор общих мультимедийных типов, которые станут более универсальными. К тому же, в чем вы вскоре сами убедитесь, мультимедийные дескрипторы обладают встроенным механизмом переключения на тип содержимого, обеспечивающим наиболее адекватное отображение информации в браузере.

Проверка поддержки в браузере

Самый простой способ тестирования поддержки дескрипторов `video` и `audio` браузером — это их динамическое создание с помощью сценария и проверка существования какой-либо из их функций, как показано ниже.

```
var hasVideo = !!(document.createElement('video').canPlayType);
```

Эта простая строка кода динамически создает элемент `video` и проверяет существование .функции `canPlayType()`. Оператор `!!` преобразует результат в булево значение, по которому можно судить, была ли попытка создания объекта `video` успешной.

Для случаев, когда необходимая поддержка отсутствует, можно предусмотреть сценарий, который вставит дескрипторы, воспринимаемые старыми браузерами, и обеспечит нужную функциональность, но с использованием таких, например, технологий, как Flash.

В элементы `audio` и `video` можно также помещать альтернативное содержимое, которое будет отображаться вместо дескрипторов, которые не поддерживаются. В простейшем случае таким альтернативным содержимым может служить замещающий текст. Это нетрудно сделать, добавив в элемент `video` или `audio` подходящее текстовое сообщение, например такое:

```
<video src="video.ogg" controls>
  Ваш браузер не поддерживает элемент "video".
</video>
```

Однако возможно и другое решение. Если браузер не поддерживает мультимедийные дескрипторы HTML5, но вы хотите отобразить видео, используя другой доступный метод, то можете предоставить в качестве содержимого неподдерживаемого элемента ссылку на внешний подключаемый модуль, который сможет воспроизвести мультимедийное содержимое.

```
<video src="video.ogg">
  <object data="videoplayer.swf" type="
    application/x-shockwave-flash">
```

```
    <param name="movie" value="video.swf"/>
  </object>
</video>
```

При наличии элемента `object`, внедренного в элемент `video`, как показано в только что приведенном примере, предпочтение будет отдаваться видео HTML5, если оно доступно, тогда как видео, воспроизводимое с помощью подключаемого модуля Flash, будет использоваться в качестве альтернативного варианта. К сожалению, до тех пор пока поддержка HTML5 не станет повсеместной, такой подход будет требовать предоставления нескольких версий одного и того же видео.

Мультимедиа для всех

Говорит Брайан: "Повышение доступности веб-приложений — это не просто разумная вещь; это неплохой бизнес, а в некоторых случаях — даже требование законодательства! Пользователи с нарушениями зрения или слуха должны получать альтернативное содержимое, которое удовлетворяло бы их потребности в получении информации.

Органам стандартизации, ответственным за разработку спецификации HTML5, известно об отсутствии в ней необходимой поддержки звука и видео (например, сопровождающих субтитров), ориентированной на людей с ограниченными возможностями, и в настоящее планируется принятие необходимых мер. А пока что разработчики должны предоставлять хотя бы ссылки на соответствующий описательный текст и предусматривать использование функций в сценариях для вывода синхронизированного текста поверх отображаемого видео или рядом с ним.

Имейте в виду, что альтернативное содержимое, размещенное внутри элементов `video` и `audio`, отображается лишь в том случае, если браузер вообще не поддерживает эти элементы. Поэтому оно не подходит в тех случаях, когда поддержка мультимедийных элементов HTML5 присутствует, но не способна удовлетворить специфические потребности пользователей.

Мультимедийные элементы

Благодаря продуманному проектному решению элементы `audio` и `video` в HTML5 во многом схожи между собой. Среди поддерживаемых ими операций есть много общих — воспроизведение, пауза, отключение/включение звука, загрузка и т.д., в связи с чем общие аспекты их поведения были выделены в отдельный раздел спецификации, посвященный *мультимедийным* элементам. Рассмотрим мультимедийные элементы более подробно, обращая внимание на то общее, что имеется между ними.

Основы: объявление мультимедийного элемента

Общие аспекты поведения мультимедийных элементов HTML5 мы рассмотрим на примере элемента `audio`. В примерах, приведенных в этом разделе, интенсивно используются мультимедийные данные, содержащиеся во вспомогательных файлах, которые вы найдете на сайте книги.

В качестве простейшего примера (файл `audio_simple.html`) рассмотрим создание страницы, отображающей аудиоплеер, способный воспроизводить приятную успокаивающую музыку — арию из сюиты для оркестра № 3 Ре мажор Иоганна Себастьяна Баха.

```
<!DOCTYPE html>
<html>
  <title>HTML5 Audio </title>
  <audio controls src="johann_sebastian_bach_air.ogg">
    Аудиоклип на основе произведения Иоганна Себастьяна Баха
  </audio>
</html>
```

Здесь предполагается, что HTML-документ и звуковой файл (в данном случае — `johann_sebastian_bach_air.ogg`) находятся в одном и том же каталоге. Как видно из рис. 3.2, при просмотре документа в браузере, поддерживающем дескриптор `audio`, отображается простой проигрыватель с временной шкалой воспроизведения, представляющей проигрываемый звуковой клип. Как и следовало ожидать, щелчок на кнопке воспроизведения приводит к проигрыванию звуковой дорожки.

Рис. 3.2. Простые элементы управления воспроизведением звука

Атрибут `controls` сообщает браузеру, что необходимо отобразить обычные пользовательские элементы управления, позволяющие запускать мультимедийный клип с самого начала или с некоторой промежуточной точки, останавливать его, а также регулировать громкость воспроизведения. Если убрать атрибут `controls`, то указанные элементы управления окажутся скрытыми, и у пользователя не будет никакой возможности начать воспроизведение клипа.

Содержимое элемента `audio`, заключенное между его открывающим и закрывающим дескрипторами, представляет собой текст, который будет отображаться в браузерах, не поддерживающих данный элемент. Это пространство также можно использовать для включения альтернативного средства визуализации мультимедийных данных, например для указания подключаемого модуля Flash или помещения прямой ссылки на мультимедийный файл.

Использование элемента `source`

Наконец, рассмотрим подробнее один из важных атрибутов — `src`. В простейших ситуациях используется один атрибут `src`, который указывает на единственный файл, содержащий звуковой клип. Но что если используемые контейнер или кодек (в нашем случае это `Ogg` и `Vorbis`) не поддерживаются данным браузером? В подобных случаях можно использовать альтернативное объявление, включающее в себя несколько источников, из которых браузер может выбрать наиболее подходящий (см. файл примера `audio_multisource.html`).

```
<audio controls>
    <source src="johann_sebastian_bach_air.ogg">
    <source src="johann_sebastian_bach_air.mp3">
    Аудиоклип на основе произведения Иоганна Себастьяна Баха
</audio>
```

В этом коде вместо атрибута `src` дескриптора `audio` использованы два новых элемента `source`. Это позволяет браузеру выбрать источник, максимально соот-

ветствующий его возможностям, и использовать этот источник для фактического воспроизведения мультимедийного клипа. Источники обрабатываются последовательно, один за другим, поэтому браузер, способный обеспечить воспроизведение нескольких из перечисленных типов источников, использует тот из них, который встретится первым.

Примечание

Помещайте в начало списка элементов `source` те мультимедийные файлы, которые обеспечивают создание наиболее комфортных условий для пользователей или меньше всего загружают сервер.

Проверив, как работает этот пример в браузере, поддерживающем первый из указанных форматов, вы не заметите никаких отличий по сравнению с предыдущим примером. Однако теперь аудиоклип можно будет воспроизвести даже в том случае, если браузер не поддерживает формат Ogg Vorbis, но поддерживает формат MP3. Вся прелесть декларативной модели состоит в том, что при написании кода, осуществляющего взаимодействие с мультимедийным файлом, вас особо не интересует, какой контейнер или кодек будет фактически использоваться. Браузер предоставляет вам унифицированный интерфейс для манипулирования мультимедийной информацией, независимо от того, какой источник соответствует возможностям воспроизведения.

Вместе с тем, существует еще одна возможность подсказать браузеру, какой источник следует использовать. Вспомните, что один медиаконтейнер может поддерживать несколько кодеков различного типа, и вам станет ясно, что браузер может быть "введен в заблуждение" при выборе подходящего типа на основании расширения объявленного файла источника. Если для атрибута `type` задается значение, не соответствующее источнику, то браузер может отказаться от воспроизведения мультимедийных данных. Поэтому будет разумно указывать тип лишь в том случае, если он точно известен. В противном случае лучше опустить этот атрибут и позволить браузеру самостоятельно определить тип кодирования. Кроме того, заметьте, что формат WebM допускает использование лишь одного аудиокодека и одного видеокодека. Это означает, что расширение имени файла `.webm` или указание `video/webm` в качестве типа содержимого сообщает браузеру все, что необходимо знать о файле. Если браузер способен воспроизводить формат `.webm`, то он сможет воспроизвести любой корректный `.webm`-файл, как видно из приведенного ниже примера кода (см. файл примера `audio_type.html`).

```
<audio controls>
    <source src="johann_sebastian_bach_air.ogg"
      type="audio/ogg; codecs=vorbis">
    <source src="johann_sebastian_bach_air.mp3" type="audio/mpeg">
    Аудиоклип на основе произведения Иоганна Себастьяна Баха
</audio>
```

Вы видите, что атрибут `type` может объявлять как тип контейнера, так и тип кодека. Использованные здесь значения представляют соответственно Ogg Vorbis и MP3. Полный список возможных типов приведен в документе RFC 4281, сопровождением которого занимается Специальная комиссия интернет-разработок (Internet Engineering Task Force, IETF); некоторые распространенные комбинации перечислены в табл. 3.2.

Таблица 3.2. Мультимедийные типы и значения атрибутов

Тип	Значение атрибута
Видео Theora и звук Vorbis в контейнере Ogg	`type='video/ogg; codecs="theora, vorbis"'`
Звук Vorbis в контейнере Ogg	`type='audio/ogg; codecs=vorbis'`
Видео H.264 Simple Baseline и звук AAC Low Complexity в контейнере MP4	`type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'`
Видео MPEG-4 Simple Profile и звук AAC Low Complexity в контейнере MP4	`type='video/mp4; codecs="mp4v.20.8, mp4a.40.2"'`

Элементы управления

Вы уже видели, что элементы управления, установленные по умолчанию, можно отобразить с помощью атрибута `controls` элемента `video` или `audio`. Как и можно было ожидать, в отсутствие этого атрибута элементы управления не будут видны в процессе отображения видео, а в случае воспроизведения звуковых файлов вообще ничего не будет отображаться, поскольку единственным визуальным представлением элемента `audio` являются его элементы управления. (В случае элемента `video` без атрибута `controls` видео по-прежнему будет отображаться.) Опускание атрибута `controls` не должно приводить к отображению какого-либо содержимого, мешающего нормальной визуализации страницы. Одним из способов вызвать воспроизведение мультимедийных данных является установка другого атрибута в медиадескрипторе: `autoplay` (см. файл примера `audio_no_control.html`).

```
<audio autoplay>
    <source src="johann_sebastian_bach_air.ogg"
      type="audio/ogg; codecs=vorbis">
    <source src="johann_sebastian_bach_air.mp3" type="audio/mpeg">
    Аудиоклип на основе произведения Иоганна Себастьяна Баха
</audio>
```

Если включен атрибут `autoplay`, то медиафайл начнет воспроизводиться сразу же после окончания его загрузки без вмешательства со стороны пользователя. Эффекты подобного рода лишь раздражают пользователей, и поэтому использование атрибута `autoplay` требует осмотрительности. Неожиданное воспроизведение музыки без предварительного предупреждения может сознательно использоваться для создания некой таинственной атмосферы или (что гораздо хуже) для навязывания рекламы пользователю. В любом случае это будет мешать воспроизведению другого звукового материала и создавать значительные помехи тем пользователям, которые осуществляют навигацию по веб-содержимому с помощью экранного диктора.

Если встроенные элементы управления не вписываются в компоновку вашего пользовательского интерфейса или управление мультимедийным элементом должно осуществляться на основе вычисляемых или поведенческих характеристик, воздействие на которые с помощью элементов управления, используемых по умолчанию, невозможно, то в JavaScript предусмотрено множество функций и атрибутов, способных помочь вам в этом. Некоторые из таких наиболее часто используемых функций перечислены в табл. 3.3.

У функции `canPlayType(type)` есть одно нетривиальное применение: передав MIME-тип произвольного видеоклипа в динамический созданный элемент `video`, можно с помощью простого сценария определить, поддерживает ли браузер данный тип. Например, приведенный ниже код позволяет быстро определить, поддерживает ли браузер MIME-тип `fooType`, не прибегая к отображению какого-либо видимого содержимого в окне браузера:

```
var supportsFooVideo =
    !!(document.createElement('video').canPlayType('fooType'));
```

Таблица 3.3. Наиболее часто используемые управляющие функции

Функция	Поведение
`load()`	Загружает медиафайл и подготавливает его к воспроизведению. Обычно необходимости в вызове этой функции не возникает, если только сам элемент не создается динамически. Используется для заблаговременной загрузки данных, подлежащих воспроизведению
`play()`	Загружает (если это необходимо) и воспроизводит медиафайл. Воспроизведение начинается с самого начала файла, если только до этого оно не было приостановлено
`pause()`	Приостанавливает воспроизведение файла, если он проигрывается в настоящее время
`canPlayType(`*тип*`)`	Проверяет, способен ли элемент `video` обеспечить воспроизведение файлов с заданным MIME-типом

В табл. 3.4 представлены некоторые атрибуты мультимедийных элементов, доступные только для чтения.

Таблица 3.4. Атрибуты мультимедийных элементов, доступные только для чтения

Атрибут	Значение
`duration`	Длительность воспроизведения всего клипа. Возвращает `NaN`, если эта характеристика неизвестна
`paused`	Возвращает `true`, если клип приостановлен. По умолчанию, если воспроизведение клипа не начиналось, возвращает `true`
`ended`	Возвращает `true`, если воспроизведение клипа завершено
`startTime`	Возвращает значение, соответствующее самому раннему моменту времени, когда может быть начато воспроизведение. Обычно это `0.0`, если только клип не является потоковым и в буфере не осталось предыдущее содержимое
`error`	Код ошибки, если она возникла
`currentSrc`	Возвращает строку, представляющую файл, который в настоящее время отображается или загружается. Это значение будет соответствовать элементу `source`, выбранному браузером

В табл. 3.5 представлены некоторые атрибуты, значения которых можно изменять в сценарии с целью непосредственного воздействия на процесс воспроизведения. В этой роли указанные атрибуты ведут себя подобно функциям.

Таблица 3.5. Атрибуты, которыми можно управлять из сценариев

Атрибут	Значение
`autoplay`	Устанавливает воспроизведение клипа сразу после его загрузки или запрашивает, установлен ли режим `autoplay`
`loop`	Возвращает `true`, если по окончании клипа он начнет воспроизводиться заново, или устанавливает для клипа режим `loop` (или его отключение)
`currentTime`	Возвращает выраженное в секундах текущее значение времени, истекшего с момента начала воспроизведения клипа. Устанавливает значение `currentTime` для перехода к определенной позиции на временной шкале воспроизведения
`controls`	Отображает или скрывает пользовательские элементы управления или запрашивает их текущее состояние видимости
`volume`	Устанавливает для уровня громкости относительное значение в интервале от `0.0` до `1.0` или запрашивает его текущее значение
`muted`	Отключает или включает звук или запрашивает текущее состояние отключения звука
`autobuffer`	Сообщает проигрывателю, следует ли загружать медиафайл, прежде чем будет инициировано его воспроизведение. Если установлен атрибут `autoplay`, этот атрибут игнорируется

Располагая богатым арсеналом функций и атрибутов, разработчик может создать пользовательский интерфейс для управления воспроизведением любого ме-

диафайла и использовать его для управления любым звуковым или видеоклипом, тип которого поддерживается браузером.

Работа со звуком

Если с атрибутами, общими для мультимедийных элементов audio и video, вам все понятно, то это означает, что о дескрипторе audio вам практически все известно. Поэтому имеет смысл сразу перейти к рассмотрению простого примера, демонстрирующего возможности воспроизведения звукового клипа из сценария.

Активизация звукового воспроизведения

Если ваш пользовательский интерфейс должен обеспечить воспроизведение звукового клипа для пользователей, но вы не хотите занимать экран временной шкалой воспроизведения или элементами управления, можете создать невидимый элемент audio (для которого атрибут controls не установлен или равен false) и предоставить собственные элементы управления воспроизведением звука. Рассмотрим следующий простой код, доступный в файле audioCue.html.

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Звуковой фон</title>

  <audio id="clickSound">
    <source src="johann_sebastian_bach_air.ogg">
    <source src="johann_sebastian_bach_air.mp3">
  </audio>

  <button id="toggle" onclick="toggleSound()">Воспроизведение</button>

  <script type="text/javascript">
    function toggleSound() {
        var music = document.getElementById("clickSound");
        var toggle = document.getElementById("toggle");

        if (music.paused) {
          music.play();
          toggle.innerHTML = "Пауза";
        }
        else {
          music.pause();
          toggle.innerHTML ="Воспроизведение";
        }
    }
  </script>
</html>
```

Здесь элемент audio вновь используется для воспроизведения полюбившейся нам мелодии Баха. Однако на этот раз пользовательские элементы управления скрыты, а автоматическое воспроизведение клипа сразу же после его загрузки отключено. Вместо этого мы создаем кнопку-переключатель, позволяющую управлять процессом воспроизведения из сценария.

```
<button id="toggle"
  onclick="toggleSound()">Воспроизведение</button>
```

Кнопка инициализируется таким образом, чтобы информировать пользователя о том, что для начала воспроизведения аудиоклипа следует щелкнуть на этой кнопке.

При каждом щелчке на кнопке запускается функция `toggleSound()`. В теле этой функции прежде всего осуществляется доступ к элементам `audio` и `button` в DOM.

```
if (music.paused) {
    music.play();
    toggle.innerHTML = "Пауза";
}
```

В результате обращения к атрибуту `paused` элемента `audio` можно проверить, приостановлено ли воспроизведение клипа пользователем. По умолчанию этот атрибут установлен в `true`, если воспроизведение не начиналось, и потому при первом щелчке это условие будет выполняться. В таком случае будет вызвана функция `play()` для данного клипа и изменен текст на кнопке, извещая пользователя о том, что следующий щелчок на ней приведет к приостановке воспроизведения клипа.

```
else {
    music.pause();
    toggle.innerHTML ="Воспроизведение";
}
```

И наоборот, если музыкальный клип не был приостановлен (воспроизводится), то воспроизведение приостановится и текст на кнопке изменится, информируя пользователя о том, что следующий щелчок приведет к возобновлению воспроизведения клипа. Все очень просто, не так ли? В этом и состоит назначение мультимедийных элементов HTML5: предложить простые способы отображения мультимедийной информации различного типа и управления ею вместо использования бесчисленного множества подключаемых модулей, как это было ранее. Простота уже сама по себе является преимуществом.

Работа с видео

Но довольно простоты! Рассмотрим нечто более сложное. Несмотря на то что элемент `video` в HTML5 во многом схож с элементом `audio`, он имеет несколько свойственных только ему атрибутов. Некоторые из них представлены в табл. 3.6.

Таблица 3.6. Дополнительные атрибуты элемента `video`

Атрибут	Значение
`poster`	URL файла изображения, используемого для представления видео до того, как оно загрузится. Считайте его своеобразной киноафишей. Этот атрибут можно считывать или устанавливать для смены изображения
`width, height`	Считывание или установка видимого размера отображаемого видео. При этом, если видео не вписывается в отведенные размеры, оно может подвергаться некоторым трансформациям
`videoWidth, videoHeight`	Возвращают внутреннюю (естественную) ширину и высоту видео. Устанавливать их значения нельзя

Элемент `video` обладает одной важной особенностью, отсутствующей у элемента `audio`: его можно передавать многим функциям HTML5 Canvas. (Более подробная информация о программном интерфейсе HTML5 Canvas содержится в главе 2.)

Создание обозревателя временной шкалы видео

В этом более сложном примере мы покажем, как можно захватить кадры элемента `video` и отобразить их на динамическом холсте. Чтобы продемонстрировать эту возможность, мы создадим простое средство просмотра временной шкалы видео. В процессе воспроизведения видео кадры из него будут периодически отображаться на расположенном рядом холсте. Если пользователь щелкнет на любом из

кадров, отображенных на холсте, воспроизведение видео возобновится с того момента времени, который соответствует данному кадру. С помощью всего лишь нескольких строк кода мы создадим обозреватель временной шкалы, который пользователи смогут использовать для перехода к нужному фрагменту продолжительного видео.

Видеоклип в нашем примере — это соблазнительная музыкальная реклама различных лакомств, которая демонстрировалась в США в середине XX столетия в кинотеатрах в перерывах между сеансами (рис. 3.3).

Рис. 3.3. Приложение, реализующее временную шкалу видео

Добавление видео и элемент `canvas`

Чтобы отобразить наш видеоклип, начнем с простого объявления:

```
<video id="movies" autoplay oncanplay="startVideo()"
    onended="stopTimeline()"
autobuffer="true" width="400px" height="300px">
    <source src="Intermission-Walk-in.ogv" type='video/ogg;
      codecs="theora, vorbis"'>
    <source src="Intermission-Walk-in_512kb.mp4"
            type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
</video>
```

Поскольку большинство элементов этой разметки вам уже знакомо из примера с аудиоклипом, остановимся на различиях. Совершенно очевидно, что элемент `audio` был заменен элементом `video`, а элементы `source` указывают на видеофильмы в форматах Ogg и MPEG, из которых браузер будет выбирать наиболее подходящий.

В данном случае для видео задан режим `autoplay`, так что воспроизведение начнется сразу после загрузки страницы. В элементе `video` зарегистрированы два обработчика событий. Когда видео загружено и готово к воспроизведению, срабатывает функция `oncanplay()`, которая запускает нашу подпрограмму. Аналогичным образом, когда видео заканчивается, функция обратного вызова `onended()` позволит нам остановить процесс создания видеокадров.

Далее мы добавляем холст с именем `timeline`, в котором через регулярные промежутки времени будут отображаться кадры видео.

```
<canvas id="timeline" width="400px" height="300px">
```

Добавление переменных

Продолжая построение нашего демонстрационного примера, начнем сценарий с объявления некоторых переменных, добавление которых облегчает настройку демонстрационного кода и повышает его удобочитаемость.

```
// временной интервал обновления кадров (в миллисекундах)
var updateInterval = 5000;

// размеры кадров, отображаемых на временной шкале
var frameWidth = 100;
var frameHeight = 75;

// количество кадров на временной шкале
var frameRows = 4;
var frameColumns = 4;
var frameGrid = frameRows * frameColumns;
```

Переменная `updateInterval` управляет частотой захвата кадров видео, которая в данном случае равна захвату одного кадра через каждые пять секунд. Переменные `frameWidth` и `frameHeight` устанавливают ширину и высоту захваченных видеокадров при их отображении на холсте в роли временной шкалы воспроизведения, а `frameRows`, `frameColumns` и `frameGrid` определяют количество кадров, отображаемых на холсте:

```
// текущий кадр
var frameCount = 0;

// для сброса таймера по завершении воспроизведения
var intervalId;

var videoStarted = false;
```

Чтобы иметь возможность отслеживать, какой именно видеокадр просматривается, введен счетчик `frameCount`, доступный всем демонстрационным функциям. (В нашем примере *кадр* — это один из образцов видео, которые захватываются каждые пять секунд.) Переменная `intervalId` используется для остановки таймера, с помощью которого отмеряется период захвата кадров. И наконец, мы добавляем флаг `videoStarted`, с помощью которого будем контролировать, чтобы для каждой демонстрации видео создавался только один таймер.

Добавление функции `updateFrame`

"Встречу" видео с холстом обеспечивает центральная функция нашего демонстрационного сценария, предназначенная для отображения захватываемых кадров видео на холсте.

```
// отображает представление видеокадра на временной шкале
function updateFrame() {
    var video = document.getElementById("movies");
    var timeline = document.getElementById("timeline");

    var ctx = timeline.getContext("2d");

    // рассчитать текущую позицию кадра на временной шкале
    // исходя из значения счетчика, а затем отобразить на ней
    // изображение, взятое из видео
    var framePosition = frameCount % frameGrid;
    var frameX = (framePosition % frameColumns) * frameWidth;
    var frameY = (Math.floor(framePosition /
```

```
                    frameRows)) * frameHeight;
    ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY,
                  frameWidth, frameHeight);

    frameCount++;
}
```

Как было показано в главе 2, первое, что необходимо сделать, приступая к работе с холстом, — это получить его графический контекст.

```
var ctx = timeline.getContext("2d");
```

Поскольку мы хотим, чтобы захватываемые кадры заполняли ячейки на холсте в направлении слева направо и сверху вниз, нам надо точно вычислять, какую именно ячейку следует использовать для очередного кадра, исходя из значения счетчика кадров. Зная ширину и высоту каждого кадра, мы можем определить точные значения координат `X` и `Y`, которые следует использовать для вывода изображения видеокадра на холсте.

```
var framePosition = frameCount % frameGrid;
var frameX = (framePosition % frameColumns) * frameWidth;
var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
```

Наконец, мы подошли к ключевому вызову, выводящему изображение на холст. Аргументы, определяющие позицию и масштаб изображения, уже встречались вам раньше, когда мы рассматривали демонстрационные примеры, иллюстрирующие работу с холстом. Однако в данном случае мы передаем функции `drawImage()` не изображение, а объект видео.

```
ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY,
  frameWidth, frameHeight);
```

Графические подпрограммы холста могут принимать источники видео в качестве изображений или шаблонов, что предоставляет весьма удобный способ изменения видео и повторного его отображения в другом месте.

Примечание

При передаче холсту видео в качестве источника входной информации он выводит лишь отображаемый в данный момент видеокадр. Выведенные на холст изображения не обновляются автоматически в процессе воспроизведения видео. Если необходимо, чтобы содержимое холста обновлялось, вы должны каждый раз перерисовывать изображения из воспроизводимого видео и заново выводить их по мере воспроизведения видео.

Добавление функции `startVideo`

Наконец, мы обновляем счетчик кадров `frameCount` всякий раз, когда захватывается очередной кадр. Теперь нам нужна только подпрограмма, регулярно обновляющая кадры для временной шкалы.

```
function startVideo() {

    // устанавливать таймер лишь при первом запуске видео
    if (videoStarted)
        return;

     videoStarted = true;

     // вычислить начальный кадр, а затем создавать
     // дополнительные кадры через регулярные промежутки
     // времени, определяемые таймером
```

```
    updateFrame();
    intervalId = setInterval(updateFrame, updateInterval);
```

Вспомните, что функция `startVideo()` запускается сразу же после загрузки видео, когда оно уже может быть воспроизведено. Прежде всего организуем проверку, гарантирующую лишь однократную обработку запуска видео для каждой загрузки страницы на тот случай, если видео перезапускается.

```
// устанавливать таймер лишь при первом запуске таймера
if (videoStarted)
    return;

videoStarted = true;
```

Захват первого кадра происходит при запуске видео. Затем мы запускаем таймер, непрерывно отмеряющий промежутки времени определенной длительности, который будет регулярно вызывать функцию `updateFrame()`. Конечный результат состоит в том, что новые кадры будут захватываться через каждые пять секунд:

```
// вычислить начальный кадр, а затем создавать
// дополнительные кадры через регулярные промежутки
// времени, определяемые таймером
updateFrame();
intervalId = setInterval(updateFrame, updateInterval);
```

Обработка пользовательского ввода

Сейчас нам осталось лишь организовать обработку щелчков, выполняемых пользователем на отдельных кадрах временной шкалы:

```
// установить обработчик для определения позиции видео,
// соответствующей кадру, на котором был выполнен щелчок
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
    var offX = evt.layerX - timeline.offsetLeft;
    var offY = evt.layerY - timeline.offsetTop;
    // вычислить, на каком кадре был выполнен щелчок,
    // при условии, что нумерация кадров начинается с нуля
    var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
    clickedFrame += Math.floor(offX / frameWidth);
    // найти соответствующий кадр в видео
    var seekedFrame = (((Math.floor(frameCount / frameGrid)) *
                                frameGrid) + clickedFrame);
    // если пользователь щелкнул на кадре, предшествующем
    // текущему кадру, то предполагается, что надо перейти
    // к последнему завершенному циклу кадров
    if (clickedFrame > (frameCount % 16))
        seekedFrame -= frameGrid;

    // нельзя перейти к позиции, предшествующей началу видео
    if (seekedFrame < 0)
        return;
```

Здесь все немного сложнее. Мы получаем холст временной шкалы и устанавливаем обработчик щелчков на нем. Обработчик будет использоваться для определения координат точки холста, в которой пользователь выполнил щелчок:

```
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
    var offX = evt.layerX - timeline.offsetLeft;
    var offY = evt.layerY - timeline.offsetTop;
```

После этого, исходя из размеров кадра, находим, на каком именно кадре из шестнадцати щелкнул пользователь.

```
// вычислить, на каком именно кадре был выполнен щелчок,
// при условии, что нумерация кадров начинается с нуля
var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
clickedFrame += Math.floor(offX / frameWidth);
```

Кадром, на котором выполняется щелчок, должен быть лишь один из самых последних кадров, поэтому определим, какой именно из недавних кадров соответствует данному индексу.

```
// найти фактический номер кадра с момента начала
// воспроизведения видео
var seekedFrame = (((Math.floor(frameCount / frameGrid)) *
                            frameGrid) + clickedFrame);
```

Если пользователь щелкнул перед текущим кадром, вернемся на один полный цикл кадров назад для определения фактического времени.

```
// если пользователь щелкнул перед текущим кадром,
// предположить, что это был последний цикл
if (clickedFrame > (frameCount % 16))
    seekedFrame -= frameGrid;
```

И наконец, мы должны предпринять меры предосторожности на тот случай, если пользователь щелкнет на кадре, который расположен до начала видеоклипа.

```
// нельзя осуществлять поиск до момента начала видео
  if (seekedFrame < 0)
      return;
```

Теперь, когда мы знаем, какому моменту времени соответствует поиск, можно использовать эти знания, чтобы изменить момент времени, являющийся текущим моментом для воспроизведения. Несмотря на то что этот этап является самым важным в нашем демонстрационном примере, реализующая его процедура довольно проста.

```
// перейти к данному кадру в видео (в секундах)
var video = document.getElementById("movies");
video.currentTime = seekedFrame * updateInterval / 1000;

// установить требуемое значение счетчика кадров
frameCount = seekedFrame;
```

Изменяя значение атрибута `currentTime` элемента `video`, мы заставляем видео вернуться к указанному моменту времени и переустановить счетчик текущих кадров на вновь выбранный кадр.

Примечание

В отличие от многих таймеров JavaScript, в которых используются миллисекунды, значение атрибута `currentTime` указывается в секундах.

Добавление функции `stopTimeline`

В нашем примере осталось лишь организовать остановку захвата кадров по окончании видео. Делать это не обязательно, но если мы не позаботимся об этом, то после того, как демонстрация видео закончится, захват кадров будет продолжаться, заполняя всю временную шкалу пустыми кадрами.

```
// остановить захват кадров
function stopTimeline() {
```

```
    clearInterval(intervalId);
}
```

Обработчик `stopTimelineHandler` запустится тогда, когда по завершении воспроизведения видео будет запущен другой обработчик — `onended`.

Возможно, наша временная шкала недостаточно функциональна для того, чтобы удовлетворить взыскательных пользователей, но для ее реализации нам потребовалось совсем немного времени. Наслаждайтесь зрелищем!

Дополнительные рекомендации

Некоторые методики, не нашедшие отражения в основных примерах данной книги, могут быть весьма полезными во многих типах приложений HTML5. Ниже представлены небольшие фрагменты кода, представляющие практический интерес, вместе с соответствующими комментариями и рекомендациями.

Фоновый звук на странице

Многие веб-сайты пытаются развлекать посетителей, по умолчанию воспроизводя звуковое сопровождение. Не являясь сторонниками такой практики, все же приведем пример того, как легко это делается средствами HTML5.

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Фоновая музыка</title>

  <audio autoplay loop>
      <source src="johann_sebastian_bach_air.ogg">
      <source src="johann_sebastian_bach_air.mp3">
  </audio

  <h1>Вам очень нравится музыка Баха!</h1>

</html>
```

Видно, что организация циклического воспроизведения звукового сопровождения сводится к объявлению единственного дескриптора `audio` с атрибутами `autoplay` и `loop` (рис. 3.4).

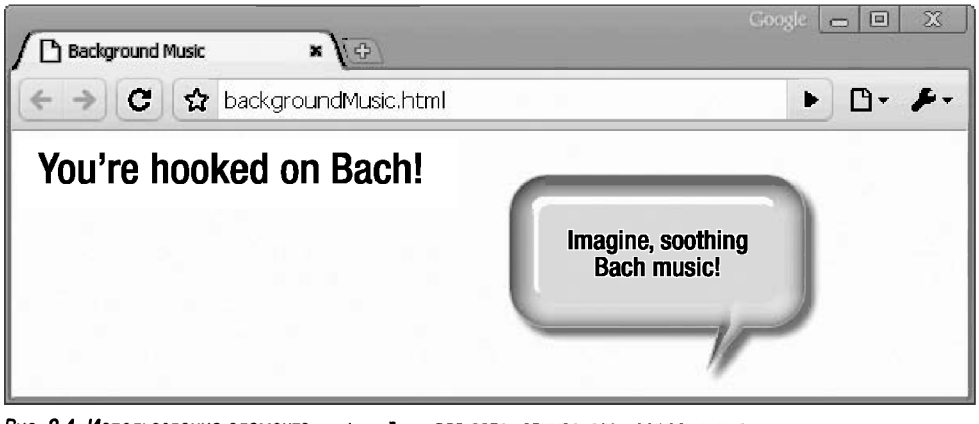


Рис. 3.4. Использование элемента `autoplay` для автоматического воспроизведения музыки при загрузке страницы

Потеря зрителей в мгновение ока

Говорит Брайан: "Большая власть означает большую ответственность, и из того факта, что вы что-то *можете*, вовсе не следует, что вы *должны* это делать. Если вам нужен пример, вспомните хотя бы пресловутый дескриптор <BLINK>!

Пусть вас не прельщает та легкость, с которой вы можете автоматически запускать воспроизведение музыки или видео, поскольку это не всегда будет уместно делать. Если у вас имеются веские причины для того, чтобы активизировать воспроизведение мультимедиа с помощью дескриптора `autoplay` (например, в случае мультимедийного браузера, когда пользователь ожидает начала воспроизведения содержимого сразу же после его загрузки), позаботьтесь о предоставлении легкодоступных средств для отключения этой возможности. Ничто не будет отталкивать пользователей от вашего сайта больше, чем раздражающая музыка, которую они не в состоянии отключить."

Воспроизведение видео при наведении на него указателя мыши

Другой простой возможностью эффективного использования сценариев для управления видеоклипами является вызов функций `play()` и `pause()` в зависимости от перемещения указателя мыши над этими элементами. Это может пригодиться на сайте, который содержит множество клипов и предоставляет пользователю возможность самостоятельно выбрать, какой из них следует воспроизвести. Видеогалерея может воспроизводить короткие видеоклипы, предназначенные для воспроизведения версий предварительного просмотра при наведении на них указателя мыши и полных версий при выполнении щелчка на них. Этот эффект довольно легко создать, используя в качестве примера приведенный ниже код (см. файл примеров `mouseoverVideo.html`).

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Эффект наведения</title>

 <video id="movies" onmouseover="this.play()" on-
mouseout="this.pause()"

autobuffer="true"
    width="400px" height="300px">
    <source src="Intermission-Walk-in.ogv" type='video/ogg;
                codecs="theora, vorbis"'>
    <source src="Intermission-Walk-in_512kb.mp4" type='video/mp4;
                codecs="avc1.42E01E, mp4a.40.2"'>
  </video>
</html>
```

В результате простой установки нескольких атрибутов мы достигаем того, что наведение указателя мыши на видео запускает его воспроизведение, как показано на рис. 3.5.

Резюме

В этой главе вы познакомились с возможностями двух очень важных элементов HTML5: `audio` и `video`. Было продемонстрировано, насколько легко с их помощью создавать профессиональные веб-приложения. Эти элементы вводят в приложения HTML5 новые возможности, позволяющие использовать звук и видео без помощи подключаемых модулей, и в то же предоставляют обычный интегрированный программный интерфейс, управляемый сценариями.

Рис. 3.5. Воспроизведение видео при наведении на него указателя мыши

Прежде всего мы обсудили контейнерные файлы и кодеки звука и видео и выяснили, почему следует постепенно отказаться от кодеков, используемых в настоящее время. Затем был описан механизм, позволяющий переключаться на тип содержимого, наиболее подходящий для конкретного браузера.

Далее было показано, как управлять аудио- и видеосодержимым программным путем с помощью встроенных функций, и, наконец, рассмотрено, как использовать программные интерфейсы HTML5 Audio и Video в собственных приложениях.

В следующей главе мы покажем, как использовать геолокацию для привязки содержимого, выводимого вашим приложением, к местонахождению пользователей, прилагая для этого лишь минимальные усилия.

Глава 4

Геолокационные средства

Предположим, вы решили создать веб-приложение, которое, например, могло бы вовремя сообщить пользователям о только что появившейся возможности приобрести кроссовки со скидкой или на льготных условиях и при этом предложить каждому из них ближайшую торговую точку, где это можно сделать, до которой он сможет пройтись пешком (или пробежаться). Используя HTML5 Geolocation API, можно запросить у пользователя разрешение на получение информации о текущем местоположении (географических координатах) браузера и в случае получения такого разрешения предоставить подробные инструкции относительно того, как добраться до магазина и совершить покупку.

В качестве еще одного примера можно привести приложение, позволяющее отслеживать, какое расстояние прошел (или пробежал) пользователь во время прогулки. Представьте, что вы используете такое приложение в браузере мобильного телефона, который включаете перед началом пробежки. Все время, пока вы бежите, приложение отслеживает покрытое вами расстояние. Получаемые координаты можно отображать на карте — возможно, даже с выводом информации о высоте текущего местоположения над уровнем моря. Если речь идет о соревнованиях, то приложение может параллельно отображать текущее местоположение соперников.

На базе HTML5 Geolocation API можно реализовать множество других, самых разнообразных идей. Так, например, браузер, опираясь на данные системы GPS, сможет подготовить для пользователя детальный маршрут следования к месту назначения или определить, где в данный момент находятся его друзья, чтобы они смогли выбрать ближайшее кафе для встречи.

В этой главе исследуются возможности программного интерфейса HTML5 Geolocation, с помощью которого пользователи могут сообщать веб-приложениям географические координаты своего текущего местоположения и тем самым извлекать все выгоды от возможности получения услуг, зависящих от этой информации. Прежде всего, мы рассмотрим те данные, которые функции HTML5 Geolocation используют для определения физического местоположения пользователя, — широту, долготу и некоторые другие величины, а также возможные источники этих данных (GPS-навигаторы, сети Wi-Fi, триангуляционные спутниковые системы и т.д.). Затем мы обсудим вопросы конфиденциальности, связанные с использованием геолокационных характеристик местонахождения пользователя, и методы обработки этих данных браузерами.

Наконец, мы вплотную займемся практическими аспектами геолокации с использованием программного интерфейса HTML5 Geolocation. В нем предусмотрено два типа запросов местоположения — однократный и многократный, требующий периодического обновления данных, и для каждого из них будут приведены соответствующие примеры их использования. Далее будет показано, как построить реальное приложение, основанное на использовании геолокационных функций, а за-

вершится наше рассмотрение обсуждением некоторых практических примеров и рекомендаций.

Информация, используемая для указания местоположения

Использовать программный интерфейс HTML5 Geolocation очень просто. Вы запрашиваете местоположение пользователя, и, если пользователь дает на это свое согласие, браузер возвращает соответствующую информацию. Данные о текущем местоположении предоставляются устройством (например, ноутбуком или мобильным телефоном), на котором установлен браузер, поддерживающий возможности геолокации. Информация о местоположении передается в виде набора координат, определяющих географическую широту и долготу, и дополнительных метаданных. Обладая этой информацией, вы можете создавать чрезвычайно интересные приложения, предлагающие новые геоинформационные сервисы.

Географические координаты — широта и долгота

Информация о местоположении в основном включает в себя пару значений географических координат — широты и долготы, наподобие тех, которые используются в следующем примере и соответствуют координатам прекрасного города Тахо, расположенного на берегу одноименного озера — самого живописного из всех горных озер Америки.

```
Широта: 39.17222, долгота: -120.13778
```

В этом примере широта (числовое значение, определяющее расстояние точки на земной поверхности от экватора в сторону севера или юга) составляет `39.17222`, а долгота (числовое значение, определяющее расстояние точки от Гринвича в сторону востока или запада) — `-120.13778`.

Широту и долготу можно задавать следующими способами:

- в виде десятичного числа (например, 39.17222);
- в формате ГМС ("градусы-минуты-секунды") (например, 39°10'20").

Пример

При работе с HTML5 Geolocation API координаты всегда возвращаются в виде десятичных градусов.

Кроме широты и долготы функции HTML5 Geolocation всегда предоставляют *точность* определения координат данного местоположения. В зависимости от устройства, на котором выполняется браузер, могут предоставляться дополнительные метаданные. К ним относятся *высота над уровнем моря* (altitude), *точность определения высоты над уровнем моря* (altitude accuracy), *курс* (heading) и *скорость* (speed). Если какое-либо из этих значений недоступно, вместо него возвращается значение `null`.

Источники информации о местоположении

Спецификация HTML5 Geolocation не содержит никаких предписаний относительно того, какую базовую технологию должно использовать устройство для определения физического местоположения пользователя. Вместо этого она ограничивается формулировкой требований к программному интерфейсу, обеспечивающе-

му извлечение этой информации. Что, однако, оговаривается в спецификации, так это предоставление точности определения координат, поскольку нет никакой гарантии того, что возвращаемые устройством координаты соответствуют его истинному местоположению.

Местоположение, местоположение...

Говорит Питер: "Вот вам один забавный пример из жизни. Дома я пользуюсь беспроводной сетью. Открыв в Firefox пример геолокационного приложения HTML5, приведенный в этой главе, я обнаружил, что, в соответствии с сообщенными приложением данными, нахожусь в Сакраменто (удаленном примерно на 75 км от места моего действительного пребывания). Ошибка? Безусловно, но меня это не особенно смутило, поскольку именно в Сакраменто находится мой интернет-провайдер (ISP).

После этого я попросил своих сыновей Шона и Рокки просмотреть ту же страницу с помощью iPod Touch (используя ту же сеть Wi-Fi). Координаты, сообщенные Safari, соответствовали Мэрисвиллю, расстояние от которого до Сакраменто составляет 30 км. Ну, и что вы на это скажете?"

Источником исходной геолокационной информации для устройства могут служить следующие данные:

- IP-адрес;
- данные триангуляционной съемки:
  - данные глобальной системы позиционирования (GPS);
  - MAC-адреса в сетях Wi-Fi, извлеченные из электронных меток RFID, Wi-Fi или Bluetooth;
  - идентификационные номера телефонов в сетях GSM и CDMA;
- данные, определенные пользователем.

Для повышения точности определении координат многие устройства используют данные, полученные из нескольких источников. У каждого из перечисленных способов есть свои преимущества и недостатки, о которых говорится в следующих разделах.

Получение геолокационной информации с помощью IP-адресов

В прошлом геолокация по IP-адресам была единственным способом определения возможного местоположения пользователя, но возвращенные результаты часто оказывались ненадежными. Геолокация этого типа основана на автоматическом распознавании IP-адреса и последующем определении физического адреса лица, на которое этот IP-адрес зарегистрирован. Поэтому, если вы пользуетесь услугами интернет-провайдера, предоставляющего вам IP-адрес, ваше местоположение часто будет разрешаться до физического адреса провайдера, который может находиться от вас на расстоянии многих километров. Преимущества и недостатки определения геолокационных данных на основании IP-адресов представлены в табл. 4.1.

Таблица 4.1. Преимущества и недостатки геолокации на основе IP-адресов

Преимущества	Недостатки
Доступность в любом месте	Невысокая точность (местоположение определяется лишь с точностью до населенного пункта, но даже в этом случае велика вероятность ошибок)
Обработка данных на стороне сервера	Стоимость может быть довольно высокой

Многие веб-сайты при публикации рекламы учитывают местонахождение пользователя. Вам может предоставиться возможность самостоятельно в этом убедиться, если во время пребывания в другой стране вы вдруг увидите на экране своего ноутбука рекламу местных услуг (зависящую от страны или региона, в котором вы в данный момент находитесь).

Получение геолокационной информации с помощью технологии GPS

Коль скоро вы видите небо, система GPS способна предоставить вам точнейшие данные о вашем местоположении. GPS определяет его на основании сигналов, получаемых от навигационных спутников, вращающихся вокруг Земли. Однако для получения результата даже однократного измерения требуется определенное время, что не всегда подходит для приложений, специфика которых требует быстрого запуска.

Поскольку процесс определения местоположения с помощью GPS может быть длительным, иногда целесообразно использовать для этого асинхронные запросы. Для индикации того, что местоположение установлено, можно использовать строку состояния. Преимущества и недостатки геолокации на основе данных GPS представлены в табл. 4.2.

Таблица 4.2. Преимущества и недостатки геолокации на основе GPS

Преимущества	Недостатки
Высокая точность	Относительно длительный процесс определения местоположения, что может приводить к быстрому истощению батареи питания пользовательского устройства
	Неустойчивая работа внутри помещений
	Может возникнуть необходимость в дополнительном оборудовании

Получение геолокационной информации с помощью технологии Wi-Fi

Сети Wi-Fi используются главным образом для определения местоположения в городских условиях путем измерения расстояний до нескольких известных точек доступа Wi-Fi. В отличие от GPS, сети Wi-Fi позволяют получать очень точные результаты как внутри, так и вне помещений.

Преимущества и недостатки геолокации на основе сетей Wi-Fi представлены в табл. 4.3.

Таблица 4.3. Преимущества и недостатки геолокации на основе сетей Wi-Fi

Преимущества	Недостатки
Точность	Не подходит для работы в условиях сельской местности с небольшим количеством точек доступа
Работает внутри помещений	
Быстрота и экономичность	

Получение геолокационной информации с помощью сетей мобильной связи

Мобильные телефоны используются для получения геолокационной информации путем измерения расстояний до нескольких базовых приемопередающих

станций. Этот метод приводит к довольно точным результатам. Его часто используют в сочетании с информацией, полученной от источников Wi-Fi и GPS.

Преимущества и недостатки геолокации на основе данных сетей мобильной связи представлены в табл. 4.4.

Таблица 4.4. Преимущества и недостатки геолокации на основе сетей мобильной связи

Преимущества	Недостатки
Хорошая точность	Требуется устройство с возможностью доступа к мобильному телефону или 3G-модем
Работает внутри помещений	Не подходит для работы в условиях сельской местности с небольшим количеством точек доступа
Быстрота и экономичность	

Определяемые пользователем геолокационные данные

Вместо того чтобы определять местоположение программным путем, можно предоставить пользователю возможность самостоятельно ввести необходимые данные. Приложение может разрешить пользователям вводить свой почтовый индекс, адрес и другую информацию. Располагая этими данными, оно может использовать их для предоставления услуг, учитывающих специфику местонахождения пользователя.

Преимущества и недостатки геолокации на основе данных, определяемых пользователем, представлены в табл. 4.5.

Таблица 4.5. Преимущества и недостатки геолокации на основе данных, определяемых пользователем

Преимущества	Недостатки
Пользователь может предоставить более точные данные, чем программные службы	Иногда этот метод приводит к большим погрешностям, особенно при смене пользователем места проживания
Обеспечивает предоставление геолокационных сервисов в любой местности	
Пользователь может ввести данные быстрее, чем они будут вычислены программным путем	

Поддержка спецификации HTML5 Geolocation браузерами

Поддержка спецификации HTML5 Geolocation браузерами еще не унифицирована, и процесс ее внедрения еще продолжается. Хорошая новость состоит в том, что спецификация HTML5 Geolocation была одним из первых средств HTML5, программа действий относительно которых была полностью принята и реализована. Разработка спецификации продвинулась достаточно далеко, и вероятность внесения в нее значительных изменений мала. Как показано в табл. 4.6, к моменту написания книги спецификация HTML5 Geolocation уже поддерживалась многими браузерами.

Из-за различий в уровне поддержки спецификации HTML5 Geolocation браузерами целесообразно выполнять соответствующую проверку, прежде чем использовать соответствующий программный интерфейс. Далее будет показано, как осуществить такую проверку программным путем.

Таблица 4.6. Поддержка спецификации HTML5 Geolocation браузерами

Браузер	Описание
Chrome	Поддерживается в Google Chrome версии 2 с установленным подключаемым модулем Gears
Firefox	Поддерживается в версиях 3.5 и выше
Internet Explorer	Поддерживается посредством подключаемого модуля Gears
Opera	Планируется включение поддержки в версии 10; на данный момент поддержка носит экспериментальный характер
Safari	Поддерживается в версии 4 для iPhone

Защита личных данных

Спецификация HTML5 Geolocation предписывает предоставление механизма, обеспечивающего конфиденциальность личных данных пользователя. Более того, информация о местоположении должна становиться доступной лишь после получения от пользователя соответствующего разрешения.

Это разумно уже само по себе, а кроме того, снижает остроту проблемы "Большого брата", которая вновь стала предметом дискуссий в связи с появлением геолокационных приложений HTML5. Однако результаты исследования возможных сценариев использования таких приложений говорят о том, что пользователь охотно поделится информацией о своем местоположении, если будет знать, что благодаря этому он вовремя узнает о возможности приобрести, скажем, кроссовки со скидкой 50% в магазине, находящемся всего лишь в нескольких кварталах от кафетерия, в котором он сейчас сидит за чашечкой кофе. Давайте посмотрим, чем обеспечивается защита личных данных пользователя в браузере и устройстве (рис. 4.1).

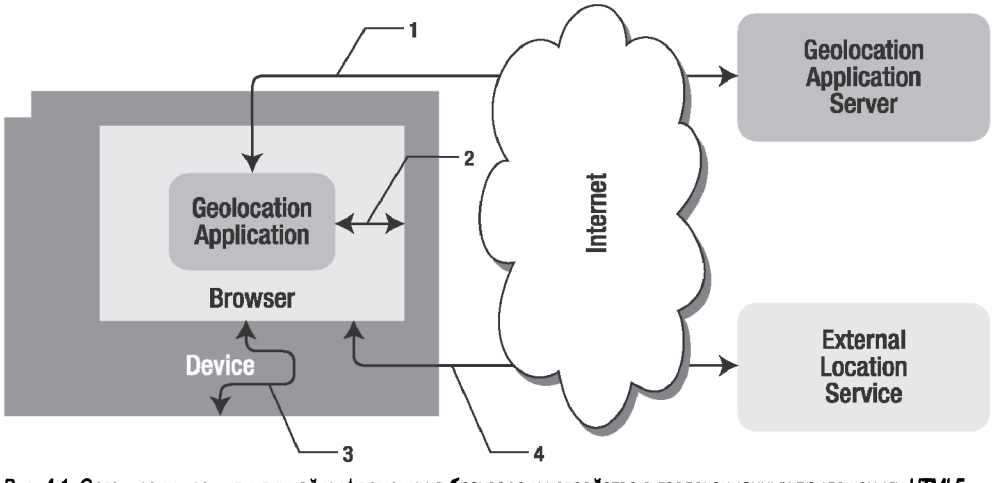


Рис. 4.1. Организация защиты личной информации в браузере и устройстве в геолокационных приложениях HTML5

Номера позиций на рисунке соответствуют следующим процессам.

1. Пользователь переходит в браузере к приложению, использующему информацию о местоположении.
2. Загружается веб-страница приложения, и с помощью вызова функции Geolocation API в браузер направляется запрос о предоставлении координат. Браузер перехватывает этот запрос и запрашивает у пользователя разреше-

ние на предоставление соответствующей информации. Предположим, что пользователь разрешает это сделать.

3. Браузер получает информацию о местоположении от устройства, на котором он выполняется. Соответствующими данными могут быть IP-адрес или координаты, найденные триангуляционным способом с использованием сетей Wi-Fi или систем GPS-навигации; это внутренняя функция браузера.
4. Браузер отсылает информацию о координатах доверенной внешней локационной службе, возвращающей детальную информацию о местоположении, которая теперь может быть отправлена обратно на хост геолокационного приложения HTML5.

Примечание

Приложение *не имеет* прямого доступа к устройству; оно лишь направляет браузеру запрос о предоставлении возможности обратиться к устройству от его имени.

Запуск механизма защиты личной информации

При попытке доступа к веб-странице, которая использует программный интерфейс HTML5 Geolocation, должен запускаться механизм защиты личной информации. Как это происходит в браузере Firefox 3.5, показано на рис. 4.2.

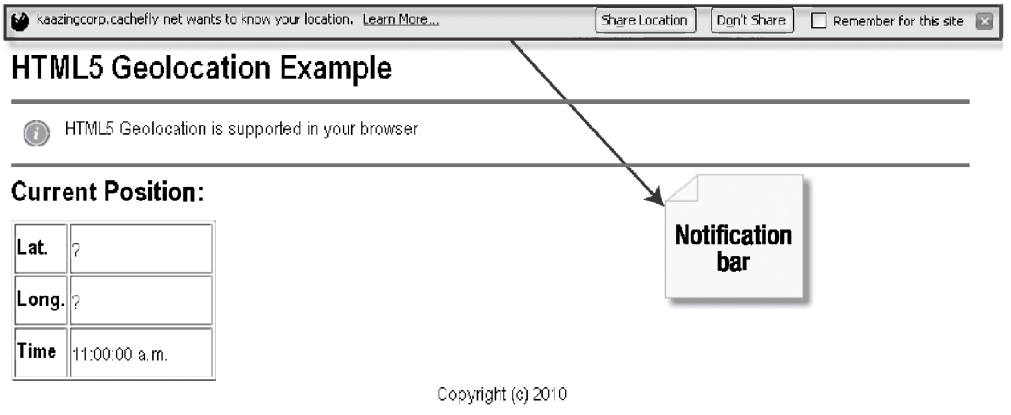


Рис. 4.2. Отображение панели уведомлений при попытке использования геолокационных функций HTML5 в браузере Firefox

Этот механизм запускается во время выполнения кода HTML5 Geolocation. Простое добавление этого кода, к которому нигде нет обращения, ни к чему не приведет. Если же код выполняется, например, при обращении к функции `navigator.geolocation.getCurrentPosition()` (о чем далее рассказывается более подробно), то пользователю будет предложено подтвердить свое согласие на предоставление приложению информации о местоположении. Как это выглядит в браузере Safari, установленном на iPhone, показано на рис. 4.3.

Кроме предоставления необходимого механизма для запроса разрешения на использование информации о местоположении, некоторые реализации (например, в Firefox) позволяют запоминать предоставление такого права сайту, чтобы использовать его для всех последующих посещений. Это аналогично возможности запоминать в браузере пароли для доступа к сайтам.

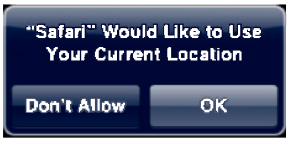


Рис. 4.3. Отображение диалогового окна запроса разрешения на использование геолокационных функций HTML5 в браузере Safari

Примечание

Если вы предоставили в Firefox какому-либо сайту постоянно действующее разрешение на получение информации о местоположении, но впоследствии решили от этого отказаться, то разрешение легко аннулировать, посетив данный веб-сайт и выбрав в меню Инструменты (Tools) пункт Информация о странице (Pafe Info). После этого следует открыть вкладку Разрешения (Permissions) и изменить настройку параметра Узнавать местоположение (Share Location).

Обработка информации о местоположении

Данные о местоположении являются критически важной информацией, и если вы их получаете, необходимо тщательно продумать способы их обработки, хранения и последующей передачи. От данных, на хранение которых не было получено специального разрешения пользователя, необходимо избавляться сразу же после того, как завершится выполнение задачи, которая их запросила.

В связи с вышеизложенным данные, подлежащие дальнейшей пересылке, рекомендуется предварительно шифровать. В зависимости от того, какие именно геолокационные данные предполагается получить, ваше приложение должно в понятной форме проинформировать пользователя о нижеследующем:

- о том, что будет осуществляться сбор данных о местоположении;
- с какой целью будет осуществляться сбор данных;
- как долго будут храниться данные о местоположении пользователя;
- какие меры предпринимаются для обеспечения секретности получаемых сведений;
- каким образом будет осуществляться общий доступ к указанным данным (если он вообще будет предоставляться);
- каким образом пользователь сможет изменить или обновить данные о своем местоположении.

Программный интерфейс HTML5 Geolocation

В этом разделе подробно рассказано о том, как использовать программный интерфейс HTML5 Geolocation. Для демонстрационных целей используется пример простой веб-страницы — `Geolocation.html`. Не забывайте, что код всех примеров доступен для загрузки с веб-страницы книги, адрес которой указан во введении.

Проверка поддержки в браузере

Прежде чем вызывать функции HTML5 Geolocation, целесообразно убедиться в том, что браузер поддерживает операции, которые вы собираетесь выполнить. Исходя из этого, вы сможете при необходимости предоставить замещающий текст, в котором пользователю приложения предлагается либо заменить устаревший

браузер, либо установить соответствующее дополнение в виде подключаемого модуля, например Gears, которое расширит существующую функциональность браузера. Один из способов, позволяющих убедиться в наличии необходимой поддержки в браузере, иллюстрирует листинг 4.1.

Листинг 4.1. Проверка необходимой поддержки в браузере

```
function loadDemo() {
  if(navigator.geolocation) {
    document.getElementById("support").innerHTML = "Ваш браузер
                          поддерживает HTML5 Geolocation API.";

  } else {
     document.getElementById("support").innerHTML = "Ваш браузер
                        не поддерживает HTML5 Geolocation API.";
  }
}
```

В этом примере тестирование поддержки осуществляется с помощью функции `loadDemo()`, которую можно вызвать при загрузке страницы приложения. Обращение к методу `navigator.geolocation()` возвращает объект `Geolocation`, если он существует; если же этот объект отсутствует, выполняется процедура обработки ошибки. Для вывода сведений о том, присутствует или не присутствует необходимая поддержка в браузере, страница обновляется путем обновления текста ранее определенного элемента `support`.

Запрос позиции

Запросы позиции бывают двух типов:

- однократный запрос;
- повторяющиеся запросы обновления.

Однократные запросы позиции

Для многих приложений вполне подойдет вариант, при котором информация о местоположении пользователя извлекается только один раз или только по запросу. Например, если кто-то захочет узнать, где находится ближайший кинотеатр, в котором через час будет демонстрироваться популярный фильм, то для этого может быть использована простейшая форма вызова, как показано в листинге 4.2.

Листинг 4.2. Однократный запрос позиции

```
void getCurrentPosition(in PositionCallback successCallback,
             in optional PositionErrorCallbackerrorCallback,
                         in optional PositionOptions options);
```

Рассмотрим этот вызов, который играет главную роль, более подробно.

Прежде всего, заметим, что эта функция доступна через объект `navigator.geolocation`, и поэтому к моменту ее вызова данный объект уже должен присутствовать в сценарии. Как отмечалось ранее, очень важно предусмотреть пути отступления на тот случай, если браузер не поддерживает спецификацию HTML5 Geolocation.

Эта функция принимает один обязательный параметр и два необязательных.

- Параметр `successCallback` сообщает браузеру, какая функция должна вызываться, когда данные о местоположении становятся доступными. Эта функция очень важна, поскольку для выполнения таких операций, как из-

влечение данных о местоположении, может потребоваться ощутимое время. Ни одному пользователю не понравится, если на протяжении всего этого времени браузер будет оставаться заблокированным, и ни один разработчик не пойдет на то, чтобы его программа простаивала в течение неопределенного времени, особенно потому, что при извлечении данных о местоположении во многих случаях надо будет дожидаться разрешения пользователя на продолжение этой операции. Именно для функции `successCallback` предназначена фактически получаемая информация о местоположении, и именно в ней она будет обрабатываться.

- Вместе с тем, как и в большинстве других сценариев, всегда имеет смысл предусмотреть план действий на случай возможных сбоев. Вполне вероятно, что запрос информации о местоположении может быть не выполнен по причинам, от вас не зависящим, так что стоит использовать функцию `errorCallback()`, которая в этом случае предоставит пользователю необходимые пояснения или, возможно, сделает повторную попытку выполнить операцию. Хотя этот параметр не является обязательным, рекомендуется предоставлять его.
- Наконец, можно указать объект `options`, позволяющий определить для службы HTML5 Geolocation детальные инструкции относительно сбора данных. Этот необязательный параметр мы рассмотрим далее.

Предположим, что обновление страницы за счет обновления данных о местоположении выполняется с помощью функции `updateLocation()`, а обработка возможных ошибок возложена на функцию `handleLocationError()`. Далее мы подробно рассмотрим структуру этих функций, а пока лишь отметим, что основной запрос, с помощью которого мы будем получать данные о местонахождении пользователя, приобретает в этом случае следующий вид:

```
navigator.geolocation.getCurrentPosition(updateLocation,
                                         handleLocationError);
```

Функция `updateLocation()`

Итак, что происходит, когда вызывается функция `updateLocation()`? На самом деле все довольно просто. Как только браузер получает доступ к информации о местоположении, он вызывает функцию `updateLocation()` с единственным параметром — объектом `position`, содержащим информацию о местонахождении (позиции) пользователя. Позиция характеризуется координатами (указанными в атрибуте `coords`) и временем определения местоположения, хранящимся во временной метке. Ключевая для определения местоположения информация содержится в атрибуте `coords`, тогда как временная метка требуется не всегда.

С координатами всегда связаны несколько атрибутов, но будут ли их значения иметь смысл, зависит от браузера и состава оборудования пользовательского устройства. Первые три атрибута таковы:

- `latitude` (широта);
- `longitude` (долгота);
- `accuracy` (точность).

Гарантируется, что эти атрибуты будут иметь определенные значения, смысл которых следует из самих названий. Атрибуты `latitude` и `longitude` будут содержать наилучшие из доступных службе HTML5 Geolocation значения координат пользователя, выраженные в десятичных градусах. Атрибут `accuracy` будет содержать выраженное в метрах значение, показывающее, насколько близко полу-

ченные значения широты и долготы соответствуют фактическому местоположению, причем уровень достоверности составляет 95%. В силу самой природы реализаций спецификации HTML5 Geolocation, точность приближения будет довольно грубой. Обязательно проверяйте точность возвращенных значений, прежде чем говорить о них с определенной степенью уверенности. Рекомендация посетить "ближайший" обувной магазин, до которого в действительности несколько часов хода, может иметь самые нежелательные последствия.

Ниже перечислены остальные атрибуты, предоставление значений которых не гарантируется, но если они не получены, то для них будут возвращаться значения `null`:

- `altitude` (высота) — высота над уровнем моря, выраженная в метрах;
- `altitudeAccuracy` (точность указания высоты) — точность, выраженная в метрах, или `null`, если высота не предоставлена;
- `heading` (курс) — направление движения, выраженное в градусном отклонении от направления на истинный север;
- `speed` (скорость) — наземная скорость, выраженная в метрах в секунду.

Если только нет уверенности в том, что пользователи имеют устройства, способные получать эту информацию, рекомендуется проектировать приложения таким образом, чтобы отсутствие любой из этих величин не могло существенно повлиять на возможность выполнения приложения. В то время как устройства глобального позиционирования способны предоставлять столь подробные геолокационные данные, триангуляционные методы этого не обеспечивают.

Перейдем к рассмотрению простой реализации кода функции `updateLocation()`, выполняющей обновление координат (листинг 4.3).

Листинг 4.3. Пример использования функции `updateLocation()`

```
function updateLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  var accuracy = position.coords.accuracy;

  document.getElementById("latitude").innerHTML = latitude;
  document.getElementById("longitude").innerHTML = longitude;
  document.getElementById("accuracy").innerHTML = "Местоположение
                определено с точностью " + accuracy + " метров."
}
```

В этом примере функция обратного вызова `updateLocation()` используется для обновления текста в трех элементах HTML-страницы; для двух из них текст элемента совпадает с его именем.

Функция `handleLocationError()`

От обработки ошибок в нашем геолокационном приложении будет зависеть очень многое, поскольку существует множество причин, из-за которых результаты, получаемые от локационных служб, могут оказаться ненадежными. К счастью, спецификация определяет коды ошибок для всех случаев, с которыми вы можете столкнуться, и устанавливает их в объекте `error`, передаваемом обработчику ошибок в качестве атрибута `code`. Подробно проанализируем каждый из этих атрибутов.

- **`UNKNOWN_ERROR`** (код ошибки `0`). Ошибка, которой не находится соответствия среди других кодов ошибки. Для получения более полной информации о природе ошибки следует обратиться к содержимому атрибута `message`.

- **PERMISSION_DENIED** (код ошибки 1). Доступ браузера к информации о местоположении не был разрешен пользователем.
- **POSITION_UNAVAILABLE** (код ошибки 2). Попытка использования данной методики определения местоположения пользователя оказалась безуспешной.
- **TIMEOUT** (код ошибки 3). Попытка определения местоположения заняла больше времени, чем допустимое время ожидания, заданное в виде параметра.

В любом из этих случаев вы, вероятно, захотите уведомить пользователя о том, что именно произошло. Также возможно, что вы захотите повторить попытку получения нужных значений в случае, если предыдущий запрос не смог дать желаемого результата, или длительность выполнения запроса превысила допустимое время ожидания.

Пример одного из возможных обработчиков ошибок приведен в листинге 4.4.

Листинг 4.4. Использование обработчика ошибок

```
function handleLocationError(error) {
    switch(error.code){
    case 0:
            updateStatus("При попытке определить ваше
                         местоположение возникла ошибка: " +
                         error.message);
            break;
    case 1:
            updateStatus("Пользователь запретил передавать
                         информацию о местоположении этой
                         странице.");
            break;
    case 2:
            updateStatus("Браузеру не удалось определить ваше
                         местоположение: " + error.message);
            break;
    case 3:
            updateStatus("Истекло допустимое время ожидания
                         информации о местоположении.");
            break;
    }
}
```

Коды ошибок извлекаются из атрибута `code`, предоставляемого объектом `error`, тогда как атрибут `message` открывает доступ к более подробному описанию причины ошибки. Во всех случаях для обновления состояния страницы новой информацией мы вызываем собственную процедуру.

Необязательные атрибуты запроса геолокационных данных

Подготовившись к нормальному выполнению программы и обработке ошибок, переключим свое внимание на три необязательных атрибута, которые можно передавать службе HTML5 Geolocation для уточнения того, как именно должны собираться данные. Обратите внимание, что эти атрибуты можно передавать с использованием сокращенной объектной нотации, что делает тривиальной задачу их добавления в вызовы, осуществляющие запрос геолокационных данных.

- `enableHighAccuracy`. Используя этот параметр, можно сообщить браузеру о желательности того, чтобы геолокационная служба использовала режим повышенной точности обнаружения, если это только возможно. По умолча-

нию этот параметр имеет значение `false`, но если он включен, то может либо не оказать никакого воздействия, либо привести к тому, что для определения местоположения устройство затратит больше времени или мощности. Этим параметром следует пользоваться с осторожностью.

Примечание

Любопытно отметить, что параметр, запрашивающий повышенную точность, является всего лишь переключателем, имеющим два значения: `true` и `false`. Спецификация не предусматривает установку различных значений или диапазона значений точности. Возможно, в будущих версиях спецификации этот недостаток будет исправлен.

- `timeout`. С помощью этого необязательного параметра, значение которого указывается в миллисекундах, можно сообщить браузеру максимально допустимое время определения местоположения. Если в течение этого времени вычисления не успеют завершиться, будет вызван обработчик ошибок. Значением по умолчанию является `Infinite`, соответствующее отсутствию каких-либо ограничений.
- `maximumAge`. Этот параметр позволяет указать, как долго прежние характеристики местоположения могут считаться актуальными, прежде чем браузер должен будет попытаться вычислить их заново. Значение этого параметра указывается в миллисекундах и по умолчанию равно 0. Нулевое значение этого параметра инструктирует браузер о необходимости непрерывного перерасчета местоположения.

Примечание

Несмотря на кажущееся сходство параметров `timeout` и `maximumAge`, они имеют разное назначение. Параметр `timeout` устанавливает *допустимую длительность* расчета местоположения, тогда как параметр `maximumAge` относится к *периодичности* выполнения таких расчетов. Если длительность одиночного вычисления местоположения превысит значение, заданное параметром `timeout`, запустится процедура обработки ошибок. Если же браузер не располагает обновленными значениями местоположения, срок существования которых меньше значения `maximumAge`, то он просто сделает попытку получить свежие данные. Некоторые значения последнего параметра имеют особый смысл: значение 0 требует непрерывного получения новых данных, тогда как значение `Infinity` указывает на то, что получение обновленных данных вообще не требуется.

Заметьте, однако, что спецификация не предоставляет возможности задавать определенную периодичность перерасчета местоположения. Эта характеристика целиком определяется конкретной реализацией браузера. Все, что мы можем сообщить браузеру, — это значение `maximumAge`, относящееся к тем значениям параметров, которые он возвращает. Фактическую же периодичность вычислений мы не можем контролировать.

Видоизменим запрос местоположения, включив в него один из необязательных параметров, используя сокращенные обозначения, как показано в следующем примере.

```
navigator.geolocation.getCurrentPosition(updateLocation,
                    handleLocationError,{timeout:10000});
```

Обновленный вызов соответствует тому, что любой запрос местоположения, занимающий более 10 секунд (10000 миллисекунд), будет запускать процедуру обработки ошибок, чему в данном случае соответствует вызов функции `handleLocationError` с кодом ошибки `TIMEOUT`.

Повторные обновления информации о местоположении

Иногда однократного получения информации о текущем местоположении оказывается недостаточно. К счастью, те, кто проектировал спецификацию Geolocation, позаботились о том, чтобы переход от приложения, выполняющего лишь однократный запрос местоположения пользователя, к приложению, выполняющему такие запросы через регулярные промежутки времени, не вызывал никаких сложностей. В действительности необходимые для этого изменения сводятся к тривиальной замене функции, выполняющей запрос, как видно из приведенного ниже примера.

```
navigator.geolocation.getCurrentPosition(updateLocation,
                                         handleLocationError);
navigator.geolocation.watchPosition(updateLocation,
                                    handleLocationError);
```

Эта простая замена приведет к тому, что вместо однократного вызова функции-обработчика `updateLocation` служба Geolocation будет повторно вызывать ее через регулярные промежутки времени. Все будет выглядеть так, будто программа *отслеживает* местоположение пользователя и сообщает обо всех его изменениях.

Зачем это может понадобиться? Представьте, например, веб-страницу, отображающую для пользователя маршрут движения по городу с указанием всех перекрестков. Другой пример — веб-страница, которая постоянно обновляется, указывая на расположение ближайших АЗС вдоль автомагистрали, по которой вы мчитесь. Такая веб-страница может даже записывать и пересылать вам данные о маршруте вашего движения, чтобы впоследствии его можно было проанализировать. Обеспечить такой сервис совсем несложно, если ваше приложение способно принимать поток обновляемых данных по мере их изменения.

Отключение обновления данных о местоположении осуществляется столь же просто. Если получение регулярно обновляемых данных о местоположении пользователя приложению больше не требуется, достаточно вызвать функцию `clearWatch()`, как показано ниже.

```
navigator.geolocation.clearWatch(watchId);
```

Эта функция информирует службу Geolocation о том, что в получении обновленных данных о местоположении пользователя больше нет необходимости. Но что такое `watchID` и откуда взялся этот параметр? По существу, это просто значение, возвращаемое вызовом функции `watchPosition()`. Оно представляет собой уникальный идентификатор запроса, отслеживающего местоположение, которое впоследствии можно использовать для прекращения выполнения этого запроса. Следовательно, если приложению необходимо прекратить получение регулярно обновляемых данных о местоположении пользователя, достаточно использовать код, показанный в листинге 4.5.

Листинг 4.5. Использование функции `watchPosition`

```
var watchId = navigator.geolocation.watchPosition(updateLocation,
                                                  handleLocationError);
// использовать обновляемые данные о местоположении

// ...

// теперь процесс получения обновляемых данных можно прекратить,
// поскольку в этом больше нет необходимости
navigator.geolocation.clearWatch(watchId);
```

Создание приложения, работающего в режиме реального времени, на основе HTML5 Geolocation API

До сих пор мы концентрировали свое внимание на однократных запросах местоположения. Давайте реально оценим разнообразие возможностей программного интерфейса HTML5 Geolocation, попытавшись использовать предлагаемые ею многократные запросы местоположения пользователя для построения небольшого, но весьма полезного приложения: веб-страницы, позволяющей отслеживать пройденное расстояние.

Если у кого-то возникает необходимость определить, какой путь был пройден за определенное время, то обычно для этого используют специальное устройство, например GPS-навигатор или шагомер. Используем возможности службы HTML5 Geolocation для создания веб-приложения, отслеживающего, какой путь прошел пользователь с момента первоначальной загрузки страницы. Разумеется, в случае настольного компьютера, постоянно находящегося в одном и том же месте, толку от этого будет мало, но такая страница идеально подойдет для миллионов мобильных телефонов, способных подключаться к Интернету и обладающих встроенной поддержкой геолокации. Достаточно направить браузер телефона на эту страницу, предоставить ей доступ к вашим геолокационным данным, и она начнет периодически собирать данные о пройденном вами пути, добавляя их в текущий счетчик суммарного расстояния (рис. 4.4).

Рис. 4.4. Пример геолокационного приложения HTML5

В этом примере используются возможности функции `watchPosition()`, обсуждавшейся нами в предыдущем разделе. Всякий раз, когда поступают данные о новой позиции, мы сравниваем их с последними известными данными и рассчитываем пройденное расстояние. Это делается с помощью формулы гаверсинуса, позволяющей вычислять расстояние между двумя точками на сфере по значениям их широты и долготы. Вид формулы гаверсинуса приведен в листинге 4.6.

Листинг 4.6. Формула гаверсинуса

$$\Delta\sigma = 2\arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1\cos\phi_2\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

Если вы попытаетесь понять, как работает эта формула, то вас постигнет горькое разочарование. Поэтому, не вдаваясь ни в какие объяснения, просто приведем реализацию формулы в виде JavaScript-сценария, который каждый сможет использовать для вычисления расстояния между двумя точками на сфере (листинг 4.7).

Листинг 4.7. JavaScript-реализация формулы гаверсинуса

```
function toRadians(degree) {
  return degree * Math.PI / 180;
}

function distance(latitude1, longitude1, latitude2, longitude2) {
  // R -- радиус Земли (км)
  var R = 6371;

  var deltaLatitude = toRadians(latitude2-latitude1);
  var deltaLongitude = toRadians(longitude2-longitude1);
  latitude1 =toRadians(latitude1);
  latitude2 =toRadians(latitude2);

  var a = Math.sin(deltaLatitude/2) *
          Math.sin(deltaLatitude/2) +
          Math.cos(latitude1) *
          Math.cos(latitude2) *
          Math.sin(deltaLongitude/2) *
          Math.sin(deltaLongitude/2);

  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
  var d = R * c;
  return d;
}
```

Если вам все же захочется узнать, как эта формула получена и как она работает, обратитесь к любому учебнику по сферической тригонометрии[1]. Для наших целей мы записали преобразование из градусов в радианы и предоставили функцию `distance()`, вычисляющую расстояние между двумя позициями, заданными своими координатами широты и долготы.

Если мы будем периодически проверять текущую позицию пользователя и вычислять расстояния, покрываемые в течение регулярных промежутков времени, то сможем получить разумное приближение для суммарной величины пройденного

[1] См. статью по адресу `http://dic.academic.ru/dic.nsf/ruwiki/1174589`. — *Примеч. ред.*

расстояния. При этом будем полагать, что на протяжении каждого временного интервала совершаемое пользователем движение является прямолинейным, но это предположение делается лишь ради упрощения нашего примера.

Создание HTML-кода

Начнем с создания HTML-кода. Для этого упражнения мы его упростим, поскольку наша главная цель — это сценарий для управления данными. Будет отображаться простая таблица, в строках которой содержатся значения широты, долготы, точности, а также метки времени (в миллисекундах), истекшего с момента последнего обновления. Кроме того, мы поместим на страницу несколько текстовых индикаторов состояния, чтобы пользователь мог видеть суммарное пройденное расстояние (листинг 4.8).

Листинг 4.8. Код для HTML-страницы счетчика расстояний

```
<h1>HTML5 Geolocation Example</h1>

<p id="status">HTML5 Geolocation is <strong>not</strong>
 supported in your browser.</p>

<h2>Current Position:</h2>
<table border="1">
<tr>
<th width="40" scope="col">Latitude</th>
<td width="114" id="latitude">?</td>
</tr>
<tr>
<td> Longitude</td>
<td id="longitude">?</td>
</tr>
<tr>
<td>Accuracy</td>
<td id="accuracy">?</td>
</tr>
<tr>
<td>Last Timestamp</td>
<td id="timestamp">?</td>
</tr>
</table>

<h4 id="currDist">Current distance traveled: 0.0 km</h4>
<h4 id="totalDist">Total distance traveled: 0.0 km</h4>
```

По умолчанию все элементы таблицы привязываются к текущему моменту времени и получают свои значения, как только в приложение начинают поступать данные.

Обработка геолокационных данных

Наш первый фрагмент JavaScript-кода будет вам знаком. Мы устанавливаем обработчик, `loadDemo()`, который выполнится сразу после завершения загрузки страницы. Этот сценарий будет проверять наличие поддержки HTML5 Geolocation в браузере и отображать результат проверки с помощью удобной вспомогательной функции, изменяющей текст сообщения в строке состояния страницы. Далее сце-

нарий запрашивает режим отслеживания местоположения пользователя, как показано в листинге 4.9.

Листинг 4.9. Добавление функции `loadDemo()`

```
var totalDistance = 0.0;
var lastLat;
var lastLong;
function updateStatus(message) {
    document.getElementById("status").innerHTML = message;
}

function loadDemo() {
  if(navigator.geolocation) {
    updateStatus("HTML5 Geolocation is supported in your browser.");
    navigator.geolocation.watchPosition(updateLocation,
                                        handleLocationError,
                                        {maximumAge:20000});
  }
}

window.addEventListener("load", loadDemo, true);
```

Для обработки ошибок будет использоваться подпрограмма, которую мы ранее обсуждали, поскольку она носит довольно общий характер и вполне может быть использована для наших целей. Эта подпрограмма проверяет полученный код ошибки и соответствующим образом изменяет текст строки состояния на странице, как показано в листинге 4.10.

Листинг 4.10. Добавление кода обработки ошибок

```
function handleLocationError(error) {
  switch (error.code) {
    case 0:
    updateStatus("There was an error while retrieving your
                 location: " + error.message);
    break;

    case 1:
    updateStatus("The user prevented this page from retrieving
                 a location.");
    break;

    case 2:
    updateStatus("The browser was unable to determine your
                 location: " + error.message);
    break;

    case 3:
    updateStatus("The browser timed out before retrieving the
                 location.");
    break;
  }
}
```

Обратите также внимание на то, что в вызове функции `watchPosition()` устанавливается значение параметра `maximumAge`: `{maximumAge:20000}`. Тем самым

мы сообщаем локационной службе о том, что не хотим использовать кешированные значения, хранящиеся более 20 секунд (20000 миллисекунд). Использование этого параметра заставит страницу регулярно обновляться; вы можете немного поэкспериментировать, самостоятельно устанавливая для этого параметра большие или меньшие значения.

Основную работу будет выполнять функция `updateLocation()`. Она обновляет страницу самыми последними значениями и рассчитывает пройденное расстояние, как показано в листинге 4.11.

Листинг 4.11. Добавление функции `updateLocation()`

```
function updateLocation(position) {
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;
        var accuracy = position.coords.accuracy;
        var timestamp = position.timestamp;

        document.getElementById("latitude").innerHTML = latitude;
        document.getElementById("longitude").innerHTML = longitude;
        document.getElementById("accuracy").innerHTML = accuracy;
        document.getElementById("timestamp").innerHTML = timestamp;
```

Как нетрудно догадаться, после получения обновленного набора координат местоположения прежде всего осуществляется запись всей информации. Сначала собираются данные о широте, долготе, точности и времени определения, а затем таблица обновляется новыми значениями.

Возможно, вы предпочтете не отображать временные метки в своем приложении. Эти метки используются здесь в виде, пригодном для компьютера, но для конечного пользователя они будут мало что значить. Вы можете либо вообще убрать их, либо разработать более дружественный по отношению к пользователю индикатор времени.

Точность задается в метрах и, на первый взгляд, ее указание представляется излишним. Однако значимость любых данных зависит от их точности. Даже если вы не захотите отображать используемую точность для пользователя, вы все равно должны учитывать ее в своем коде. Предоставление пользователю неточных значений может ввести его в заблуждение относительно его действительного места нахождения. Поэтому мы будем отбрасывать любые обновленные данные, точность которых недопустимо низка, как показано в листинге 4.12.

Листинг 4.12. Игнорирование обновлений, не обладающих достаточной точностью

```
        // Вспомогательная проверка. Рассчитывать расстояние
        // при низкой точности не имеет смысла.
        // Погрешность слишком велика.
        if (accuracy >= 500) {
            updateStatus("Need more accurate values to calculate_
                          distance.");
            return;
}
```

Простейший способ путешествовать

Говорит Брайан: "Отслеживание точности определения позиции является настоятельной необходимостью. Как разработчик, вы не будете иметь доступа к методологии, используемой браузером для вычисления местоположения, но в вашем распоряжении будет атрибут точности. Пользуйтесь им!

Как то раз, отдыхая в жаркий полдень в своем гамаке, я решил понаблюдать за результатами определения своего местоположения в браузере мобильного телефона, поддерживающего геолокацию. К своему удивлению, я обнаружил, что, в течение всего лишь нескольких минут мое бренное тело, в соответствии с полученными результатами наблюдений, пропутешествовало в пространстве около полукилометра с различной скоростью. Каким бы поразительным ни казался этот результат, он еще раз напоминает о том, что данные точны настолько, насколько это обеспечивается их источником."

Наконец, мы рассчитываем пройденное расстояние в предположении, что перед этим нами уже было получено хотя бы одно точное значение позиции. Далее мы обновляем суммарную величину пройденного расстояния и отображаем ее для пользователя, сохраняя текущие значения для последующих сравнений. Чтобы излишне не загромождать интерфейс, имеет смысл округлить или обрезать рассчитанные значения, как показано в листинге 4.13.

Листинг 4.13. Добавление кода для вычисления расстояний

```
// рассчитать расстояние

        if ((lastLat !== null) && (lastLong !== null)) {
            var currentDistance = distance(latitude, longitude,
                                  lastLat, lastLong);
            document.getElementById("currDist").innerHTML =
              "Current distance traveled: " +
               currentDistance.toFixed(4) + " km";

            totalDistance += currentDistance;

            document.getElementById("totalDist").innerHTML =
              "Total distance traveled: " +
               totalDistance.toFixed(4) + " km";
        }


        lastLat = latitude;
        lastLong = longitude;

        updateStatus("Location successfully updated.");
}
```

Вот и все. Ценой всего лишь примерно 200 строк кода, реализующего HTML-страницу и сценарий, нам удалось создать пример приложения, отслеживающего изменение позиции наблюдателя с течением времени, и продемонстрировать почти все возможности программного интерфейса HTML5 Geolocation, включая обработку ошибок. В то время как этот пример не очень годится для демонстрации на настольном компьютере, вы можете испытать его на мобильном телефоне или переносном устройстве, пригодном для работы с геолокационной службой, и оценить, насколько вы действительно "подвижны" в течение дня.

Окончательный код

Полный код примера представлен в листинге 4.14.

Листинг 4.14. Завершенный код приложения для отслеживания пройденного пути

```
<!DOCTYPE html>
<html>
```

```
<title>HTML5 Geolocation Odometer</title>
<h1>HTML5 Geolocation Distance Tracker</h1>

<p id="status">HTML5 Geolocation is <strong>not</strong>
 supported in your browser.</p>

<h2>Current Position:</h2>
<table border="1">
<tr>
<th width="40" scope="col">Latitude</th>
<td width="114" id="latitude">?</td>
</tr>
<tr>
<td> Longitude</td>
<td id="longitude">?</td>
</tr>
<tr>
<td>Accuracy</td>
<td id="accuracy">?</td>
</tr>
<tr>
<td>Last Timestamp</td>
<td id="timestamp">?</td>
</tr>
</table>

<h4 id="currDist">Current distance traveled: 0.0 km</h4>
<h4 id="totalDist">Total distance traveled: 0.0 km</h4>

<script type="text/javascript">

    var totalDistance = 0.0;
    var lastLat;
    var lastLong;

    function toRadians(degree) {
      return degree * Math.PI / 180;
    }

    function distance(latitude1, longitude1, latitude2, longitude2) {
      // R -- радиус Земли (км)
      var R = 6371;
      var deltaLatitude = toRadians(latitude2-latitude1);
      var deltaLongitude = toRadians(longitude2-longitude1);
      latitude1 = toRadians(latitude1);
      latitude2 = toRadians(latitude2);

      var a = Math.sin(deltaLatitude/2) *
      Math.sin(deltaLatitude/2) +
      Math.cos(latitude1) *
      Math.cos(latitude2) *
      Math.sin(deltaLongitude/2) *
      Math.sin(deltaLongitude/2);

      var c = 2 * Math.atan2(Math.sqrt(a),
      Math.sqrt(1-a));
      var d = R * c;
      return d;
```

```
    }

    function updateStatus(message) {
        document.getElementById("status").innerHTML = message;
    }

    function loadDemo() {
        if(navigator.geolocation) {
            updateStatus("HTML5 Geolocation is supported in your
                          browser.");
            navigator.geolocation.watchPosition(updateLocation,
                                                handleLocationError,
                                                {maximumAge:20000});
        }
    }

    window.addEventListener("load", loadDemo, true);

    function updateLocation(position) {
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;
        var accuracy = position.coords.accuracy;
        var timestamp = position.timestamp;

        document.getElementById("latitude").innerHTML = latitude;
        document.getElementById("longitude").innerHTML = longitude;
        document.getElementById("accuracy").innerHTML = accuracy;
        document.getElementById("timestamp").innerHTML = timestamp;

        // проверка корректности данных... не вычислять
        // расстояние при слишком большой погрешности
        if (accuracy >= 500) {
            updateStatus("Need more accurate values to calculate
                          distance.");
            return;
        }
// вычислить расстояние

        if ((lastLat !== null) && (lastLong !== null)) {
            var currentDistance = distance(latitude, longitude,
                                           lastLat, lastLong);
            document.getElementById("currDist").innerHTML =
              "Current distance traveled: " +
               currentDistance.toFixed(4) + " km";

            totalDistance += currentDistance;

            document.getElementById("totalDist").innerHTML =
              "Total distance traveled: " +
              totalDistance.toFixed(4) + " km";
        }


        lastLat = latitude;
        lastLong = longitude;

        updateStatus("Location successfully updated.");
```

```
    }

    function handleLocationError(error) {
      switch (error.code) {
        case 0:
        updateStatus("There was an error while retrieving your
                     location: " + error.message);
        break;

        case 1:
        updateStatus("The user prevented this page from retrieving
                     a location.");
        break;

        case 2:
        updateStatus("The browser was unable to determine your
                     location: " + error.message);
        break;

        case 3:
        updateStatus("The browser timed out before retrieving the
                     location.");
        break;
      }
    }
</script>
</html>
```

Дополнительные рекомендации

Некоторые методики, не нашедшие отражения в основных примерах, приведенных в данной книге, могут быть весьма полезными во многих типах приложений HTML5. Ниже представлены небольшие фрагменты кода, представляющие практический интерес, вместе с соответствующими комментариями и рекомендациями.

Каково состояние приложения

Возможно, вы заметили, что значительная часть функций HTML5 Geolocation API связана с измерением временных характеристик. Это не должно особенно удивлять. Для получения результатов с помощью любой из методик определения местоположения — посредством использования мобильных телефонов, триангуляционных методов, GPS-навигаторов, просмотра IP-адресов, — требуется довольно длительное время, если эти результаты вообще могут быть получены. К счастью, спецификация предоставляет в распоряжение разработчика массу информации, пригодной для вывода в строке состояния полезных данных.

Если разработчик задает для процедуры определения местоположения использование необязательного параметра `timeout`, то тем самым он требует, чтобы геолокационная служба высылала ему уведомления об ошибке в тех случаях, когда для нахождения местоположения требуется больше времени, чем то, которое установлено этим параметром. Это подсказывает нам, что будет весьма разумно отображать в интерфейсе строку состояния, информирующую пользователя о ходе выполнения запроса. Начало вывода строки состояния соответствует моменту отправки запроса, а его окончание — моменту времени, определяемому значением

параметра timeout, независимо от того, удачным или неудачным окажется завершение запроса.

В листинге 4.15 представлен код запуска таймера, ответственного за регулярное обновление строки состояния новым значением индикатора хода процесса.

Листинг 4.15. Добавление строки состояния

```
function updateStatus(message) {
    document.getElementById("status").innerHTML = message;
}

function endRequest() {
    updateStatus("Done.");
}

function updateLocation(position) {
    endRequest();
    // обработать информацию о местоположении
}

function handleLocationError(error) {
    endRequest();

    // обработать возможные ошибки
}

navigator.geolocation.getCurrentPosition(updateLocation,
                                         handleLocationError,
                                         {timeout:10000});
                                      // время ожидания -- 10 секунд

updateStatus("Requesting location data...");
```

Разберем этот код по частям. Как и до этого, у нас имеется функция, предназначенная для обновления строки состояния на странице, как показано ниже.

```
function updateStatus(message) {
    document.getElementById("status").innerHTML = message;
}
```

Информация о состоянии приложения предоставляется в виде простого текста (листинг 4.16), хотя ничто не мешает вывести строку состояния в привлекательной графической форме.

Листинг 4.16. Отображение состояния приложения

```
navigator.geolocation.getCurrentPosition(updateLocation,
                                         handleLocationError,
                                         {timeout:10000});
                                      // время ожидания -- 10 секунд
updateStatus("Requesting location data...");
```

Здесь вновь для определения текущего местоположения пользователя используется программный интерфейс HTML5 Geolocation, но значение параметра timeout соответствует 10 секундам. Учитывая это, по истечении 10 секунд запрос либо окажется успешным, либо завершится с ошибкой.

Отображаемый в строке состояния текст непрерывно обновляется, показывая, что запрос находится в состоянии выполнения. Сразу после завершении запроса

или по истечении 10 секунд — в зависимости от того, какое из этих событий наступит раньше, — функция обратного вызова обновляет текст в строке состояния, как показано в листинге 4.17.

Листинг 4.17. Обновление текста в строке состояния

```
function endRequest() {
    updateStatus("Done.");
}
function updateLocation(position) {
    endRequest();
    // обработать информацию о местоположении
}
```

Этот вспомогательный код весьма прост, но легко расширяем.

Данная методика хорошо подходит для однократных измерений позиции, поскольку для разработчика не составляет труда определить момент начала запроса. Конечно же, выполнение запроса начинается сразу после вызова функции `getCurrentPosition()`. Однако в случае многократных запросов определения позиции, запускаемых с помощью функции `watchPosition()`, разработчик не в состоянии контролировать, когда именно начинается выполнение отдельных запросов.

Кроме того, отсчет интервала ожидания не начнется до тех пор, пока пользователь не разрешит геолокационной службе доступ к данным о его местоположении. Поэтому с практической точки зрения в реализации строки состояния, предоставляющей точную информацию о ходе выполнения запроса, нет никакого смысла, поскольку страница не получает никакой информации в течение того промежутка времени, когда пользователь дает свое согласие на предоставление геолокационных данных.

Как найти себя на карте Google

Весьма распространены запросы геолокационных данных, целью которых является отображение позиции пользователя на карте, как это, например, делает популярная служба Google Maps. В действительности эта служба стала настолько популярной, что компания Google по собственной инициативе встроила поддержку HTML5 Geolocation в ее пользовательский интерфейс. Щелкните на кнопке Показать мое местоположение (Show My Location) (рис. 4.5). Чтобы определить ваше местоположение и отобразить его на карте, приложение Google Maps воспользуется программным интерфейсом Geolocation (если он доступен).

Однако то же самое можно сделать и самостоятельно. Несмотря на то что программный интерфейс Google Maps не является предметом рассмотрения данной книги, следует отметить, что он предусматривает (и не случайно) использование десятичных значений широты и долготы местоположения. Поэтому не составляет труда передать результаты работы вашего приложения объекту Google Maps, как показано в листинге 4.18.

Листинг 4.18. Передача информации о местоположении объекту Google Maps

```
// Создание карты Google.
// Подробности см. в документации Google.
var map = new google.maps.Map(document.getElementById("map"));

function updateLocation(position) {
  // Передать местоположение объекту Google Maps
  // и отобразить его в центре карты
```

```
  map.setCenter(new google.maps.LatLng(position.coords.latitude,
                                       position.coords.longitude);
}
```

Рис. 4.5. Кнопка Показать мое местоположение на карте Google

Резюме

В этой главе обсуждался программный интерфейс HTML5 Geolocation. Вы узнали о том, из каких элементов формируется информация о местоположении — широта, долгота и другие атрибуты, — а также о том, откуда берутся эти данные. Вы также узнали, какие меры предусмотрены в спецификации HTML5 Geolocation для защиты личной информации пользователя, и увидели, какие замечательные приложения, основанные на использовании геолокационных данных, позволяет создавать эта спецификация.

В следующей главе будет продемонстрировано, как HTML5 позволяет организовать обмен данными между вкладками и окнами, а также между страницами и серверами с различными доменными именами.

Глава 5

Коммуникационные средства

В этой главе исследуются возможности двух важнейших компонентов технологии *кросссдоменного обмена сообщениями* — Cross Document Messaging и XMLHttpRequest Level 2 — и демонстрируется их применение для создания мощных приложений. Обе спецификации обогащают приложения HTML5 новыми возможностями и позволяют приложениям из разных источников (доменов) безопасно обмениваться сообщениями между собой.

Сначала мы обсудим два ключевых элемента, на которых базируется обмен сообщениями в HTML5, а именно: метод `postMessage()` и концепцию безопасности, основанную на понятии *источника* (origin), а затем покажем, как использовать метод `postMessage()` для обмена сообщениями между фреймами, вкладками и окнами.

Далее мы рассмотрим спецификацию XMLHttpRequest Level 2 — улучшенную версию спецификации XMLHttpRequest. Мы расскажем о том, в чем состоит суть внесенных улучшений. Кроме того, будет показано, как организовать запрос ресурсов из другого источника с помощью XMLHttpRequest и как использовать новые события `progress`.

Обмен сообщениями между документами

До недавнего времени обмен сообщениями между фреймами, вкладками и окнами в браузере полностью ограничивался из соображений безопасности. Несмотря на то что для некоторых сайтов возможность предоставления браузером содержимого веб-страницы для совместного использования была бы желательной, это могло бы создавать дополнительные лазейки для атак злоумышленников. Если бы браузеры разрешали программный доступ к содержимому, загруженному в другие фреймы и вкладки, то у сайтов появилась бы возможность незаконно извлекать информацию из содержимого других сайтов с помощью сценариев. Поэтому производители браузеров были совершенно правы, ограничивая подобный вид доступа. Попытки извлечения или изменения содержимого, загруженного из другого источника, приводят к генерированию исключений безопасности, делающих подобные операции невозможными.

Однако даже в обычной практике возможны случаи, когда контентные элементы, источниками которых являются разные сайты, должны иметь возможность взаимодействовать внутри браузера. Классическим примером могут служить так называемые "мэшапы" (mashup) — составные веб-приложения, объединяющие в себе сразу несколько приложений (например, карту, чат и новости) из различных источников в одно интегрированное мета-приложение. Такие скоординированные наборы приложений, внедренные в страницу, должны обслуживаться через прямые каналы передачи сообщений внутри самого браузера.

Для удовлетворения этих потребностей производители браузеров и органы стандартизации договорились о введении новой спецификации — средства обмена сообщениями между документами (Cross Document Messaging). Это средство обеспечивает безопасный обмен сообщениями через границы фреймов, вкладок и окон. В качестве стандартного способа отправки сообщений спецификация определяет метод `postMessage()`. Приведенный ниже пример демонстрирует простоту отправки сообщений.

```
chatFrame.contentWindow.postMessage('Hello, world',
                                    'http://www.example.com/');
```

Для получения сообщений вы должны всего лишь добавить на страницу обработчик событий. При получении сообщения можно проверить его источник и на основании этого решить, каковы должны быть дальнейшие действия. Листинг 5.1 демонстрирует использование метода `addEventListener()` для регистрации функции `messageHandler()` в качестве обработчика событий `message`.

Листинг 5.1. Добавление обработчика событий `message`

```
window.addEventListener("message", messageHandler, true);
function messageHandler(e) {
    switch(e.origin) {
      case "friend.example.com":
      // обработать сообщение
      processMessage(e.data);
      break;
    default:
      // источник сообщения не распознан;
      // игнорировать сообщение
    }
}
```

Событие `message` является событием DOM, следовательно, у объекта этого события есть свойства `data` и `origin`. Свойство `data` содержит само сообщение, переданное отправителем, а свойство `origin` — источник отправителя. Используя свойство `origin`, получатель может легко отличить сообщения, поступившие от источников, не являющихся доверенными, и проигнорировать их; для этого следует всего лишь свериться со списком доверенных источников.

На рис. 5.1 показан пример, в котором метод `postMessage()` обеспечивает обмен сообщениями между виджетом чата во фрейме с адресом хоста `http://chat.example.com` и HTML-страницей, содержащей этот фрейм, с адресом хоста `http://portal.example.com` (два разных источника).

В этом примере виджет содержится во фрейме (iframe), поэтому у него нет прямого доступа к родительскому окну. Получив сообщение чата, виджет может с помощью метода `postMessage()` известить об этом основную страницу, чтобы та могла предупредить пользователя о получении нового сообщения. Точно так же основная страница может отправлять сообщения о статусе пользователя виджету чата. Как страница, так и виджет могут обеспечить беспрепятственный обмен сообщениями между собой, добавив адреса соответствующих сайтов в "белый список" разрешенных источников.

До появления метода `postMessage()` обмен сообщениями между фреймами иногда можно было осуществить с помощью сценариев. Сценарий, выполняющийся на одной странице, пытался манипулировать другим документом. Это не всегда могло быть разрешено из соображений безопасности. Вместо использовавшегося в таких случаях прямого программного доступа метод `postMessage()` обеспечивает асин-

хронную передачу сообщений между двумя контекстами JavaScript. Как видно из рис. 5.2, попытка обмена сообщениями между разными источниками без использования метода `postMessage()` приводит к ошибкам безопасности, генерируемым браузерами во избежание атак посредством *межсайтового скриптинга* (cross-site scripting attacs).

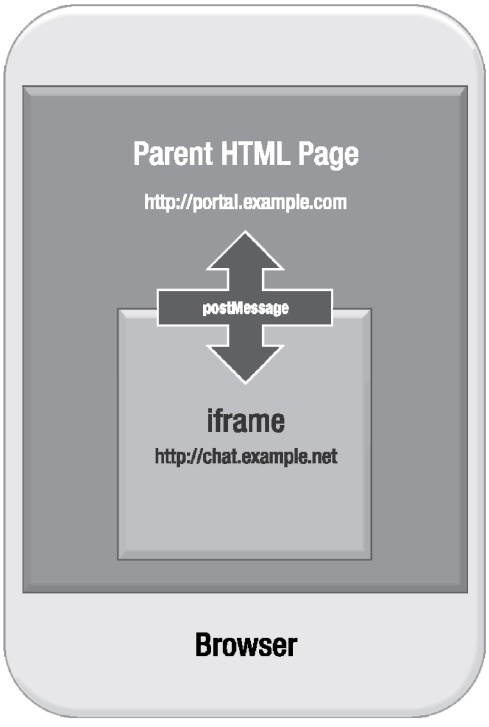


Рис. 5.1. Обмен сообщениями между фреймом и основной HTML-страницей с помощью метода `postMessage()`

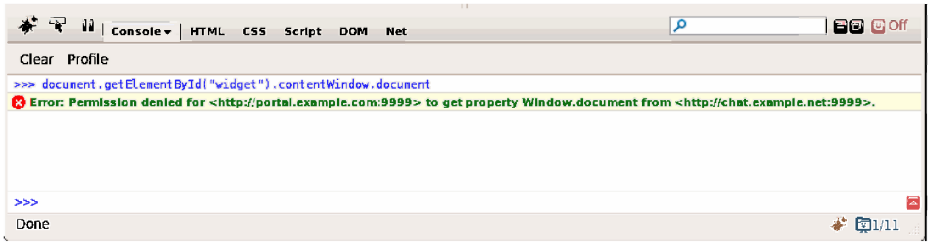


Рис. 5.2. Возникновение ошибки при попытке межсайтового скриптинга в ранних версиях Firefox и Firebug

Метод `postMessage()` может использоваться для обмена сообщениями между документами, принадлежащими одному и тому же источнику, но особенно полезным он оказывается в тех случаях, когда обмен сообщениями запрещен *политикой ограничения домена* (same origin policy), устанавливаемой браузерами. Тем не менее метод `postMessage()` стоит использовать и для обмена сообщениями в пределах одного домена, поскольку он предоставляет единообразный и простой в использовании

программный интерфейс. Этот метод используется во всех случаях, когда необходимо организовать обмен сообщениями между контекстами JavaScript, например в случае потоков выполнения HTML5 Web Workers.

Безопасность источников

HTML5 уточняет и дополняет принципы безопасности доменов, вводя понятие *источника* (origin). Источник — это часть адреса, используемая для моделирования доверительных отношений в Интернете. Источник определяется сочетанием схемы, хоста и порта. Например, страницы `https://www.example.com/` и `http://www.example.com/` относятся к разным источникам, поскольку отличаются схемами (протоколами) (`https` и `http`). Путь в источнике не принимается в расчет, и поэтому страницы `http://www.example.com/index.html` и `http://www.example.com/page2.html` имеют один и тот же источник, поскольку отличаются лишь путями.

HTML5 определяет сериализацию источников. На источники в строковой форме можно ссылаться в API-функциях и протоколах. Это имеет существенное значение при межсайтовых HTTP-запросах, выполняемых как с помощью XMLHttpRequest, так и с помощью веб-сокетов.

При обмене сообщениями между источниками отправитель идентифицируется по его источнику. Это позволяет получателю игнорировать сообщения, поступающие от источников, которым он не доверяет или от которых не ожидает получения сообщений. Более того, приложения должны регистрировать обработчики событий `message`. Поэтому риск того, что сообщения могут каким-то образом помешать работе приложений, не ожидающих их получения, полностью исключается.

Установленные для метода `postMessage()` правила безопасности гарантируют, что сообщения не будут получены страницами, источник которых неизвестен или не вызывает доверия. Посылая сообщение, отправитель указывает источник получателя. Если окно, для которого отправитель сообщения вызывает метод `postMessage()`, относится к источнику, не соответствующему указанному (например, если пользователь перешел на другой сайт), то браузер не передаст это сообщение.

Точно так же, в получаемых сообщениях содержится информация об источнике отправителя. Источник сообщения предоставляется браузером и не может быть подделан. Благодаря этому получатель имеет возможность решать, какие сообщения обрабатывать, а какие — игнорировать. Можно вести "белый" список и обрабатывать лишь сообщения, отправленные документами, источнику которых вы доверяете.

Осторожно обращайтесь с внешними данными!

Говорит Фрэнк: "В приложениях, которые обрабатывают сообщения, поступающие из других источников, всегда должна выполняться верификация источника каждого сообщения. Данные, приходящие с такими сообщения, также требуют осторожного обращения. Даже если сообщение поступило от доверенного источника, по отношению к нему следует соблюдать те же меры предосторожности, что и по отношению к любому другому внешнему источнику данных. Следующие два примера демонстрируют вставку содержимого, создающую угрозу безопасности, и безопасный альтернативный вариант.

```
// Опасно: e.data рассматривается как разметка!
element.innerHTML = e.data;

// Лучше
element.textContent = e.data;
```

Лучше всего *никогда* не анализировать строки, полученные от посторонних источников. Также избегайте использования оператора `eval` по отношению к строкам, источником происхождения которых является ваше собственное приложение. Безопаснее использовать JSON с помощью вызова `window.JSON()`

или посредством синтаксического анализатора, доступного на сайте `json.org`. JSON — это язык описания данных, предназначенный для того, чтобы обеспечить безопасное использование данных в JavaScript, а синтаксический анализатор `json.org` известен своей нетерпимостью к самым незначительным ошибкам."

Браузерная поддержка обмена сообщениями между документами

Как видно из табл. 5.1, на момент написания книги спецификация HTML5 Cross Document Messaging уже поддерживалась многими браузерами.

Таблица 5.1. Поддержка спецификации HTML5 Cross Document Messaging браузерами

Браузер	Описание
Chrome	Поддерживается в версии 2.0 и выше
Firefox	Поддерживается в версии 3.0 и выше
Internet Explorer	Поддерживается в версии 8.0 и выше
Opera	Поддерживается в версии 9.6 и выше
Safari	Поддерживается в версии 4.0 и выше

Прежде чем использовать спецификацию HTML5 Cross Document Messaging, следует убедиться в том, что она поддерживается браузером. О том, как выполнить такую проверку программным путем, говорится далее.

Использование метода `postMessage()`

Этот раздел посвящен более подробному изучению метода `postMessage()`.

Проверка поддержки в браузере

Прежде чем вызывать метод `postMessage()`, следует проверить, поддерживается ли он браузером. Ниже продемонстрирован один из возможных способов выполнения такой проверки.

```
if (typeof window.postMessage === "undefined") {
    // Метод postMessage не поддерживается этим браузером
}
```

Отправка сообщений

Чтобы отправить сообщение, необходимо вызвать метод `postMessage()` из целевого объекта `window`, как показано ниже.

```
window.postMessage("Hello, world", "portal.example.com");
```

Первый аргумент содержит отправляемые данные, а второй указывает получателя сообщения. Для отправки сообщения фрейму можно вызвать метод `postMessage()` из объекта `contentWindow`, в котором содержится фрейм, как показано в следующем примере.

```
document.getElementsByTagName("iframe")[0].contentWindow.
        postMessage("Hello, world","chat.example.net");
```

Обработка событий `message`

Для получения сообщений сценарий должен зарегистрировать обработчик событий `message` в объекте `window`, как показано в листинге 5.2. Решение о том, получить или проигнорировать сообщение, приложение-получатель может принимать в обработчике событий.

Листинг 5.2. Обработка событий сообщений и сверка источников с "белым" списком

```
var originWhiteList = ["portal.example.com", "games.example.com",
                       "www.example.com"];

function checkWhiteList(origin) {
    for (var i=0; i<originWhiteList.length; i++) {
        if (origin === originWhiteList[i]) {
          return true;
        }
    }
    return false;
}

function messageHandler(e) {
    if(checkWhiteList(e.origin)) {
        processMessage(e.data);
    } else {
        // Игнорировать сообщения из неизвестных источников
    }
}
window.addEventListener("message", messageHandler, true);
```

Примечание

Определенный в HTML5 интерфейс MessageEvent также является частью спецификаций HTML5 WebSocket и HTML5 Web Workers. Коммуникационные средства HTML5 имеют унифицированный программный интерфейс для получения сообщений. Другие коммуникационные средства, такие как EventSource, также используют интерфейс MessageEvent для отправки сообщений.

Создание приложения, использующего метод `postMessage()`

Чтобы вы могли закрепить свои знания, создадим приложение-портал с виджетом чата. Для организации связи между страницей портала и виджетом воспользуемся возможностями средства обмена сообщениями между документами. Внешний вид нашего будущего приложения иллюстрирует рис. 5.3.

В этом примере будет продемонстрировано, как портал может внедрять сторонние виджеты во фреймы. Мы ограничимся только одним виджетом, источником которого является хост `chat.example.net`. Связь между страницей портала и виджетом будет осуществляться с помощью метода `postMessage()`. В данном случае фрейм представляет виджет чата, уведомляющий пользователя о получении сообщения миганием титульного названия. Такой прием довольно часто используется в приложениях, ожидающих получения событий в фоновом режиме. Однако ввиду того, что виджет изолирован во фрейме, обслуживаемом не из родительской страницы, а из другого источника, любая попытка изменения названия в обычных условиях была бы пресечена из соображений безопасности. Суть нашего решения состоит в том, что виджет, используя метод `postMessage()`, будет посылать запросы на выполнение родительской страницей всех необходимых действий от его имени.

В данном примере портал также будет отправлять фрейму сообщения, информирующие виджет об изменении статуса пользователя. Таким образом, использование метода `postMessage()` позволяет порталам, подобным нашему, координировать свои действия с виджетами в рамках "составного" приложения. Разумеется,

благодаря проверке, которой подвергается как целевой источник при отправке сообщения, так и источник получаемого события, вероятность случайного получения данных или их передачи через соединение, установленное путем подделки адреса, исключается.

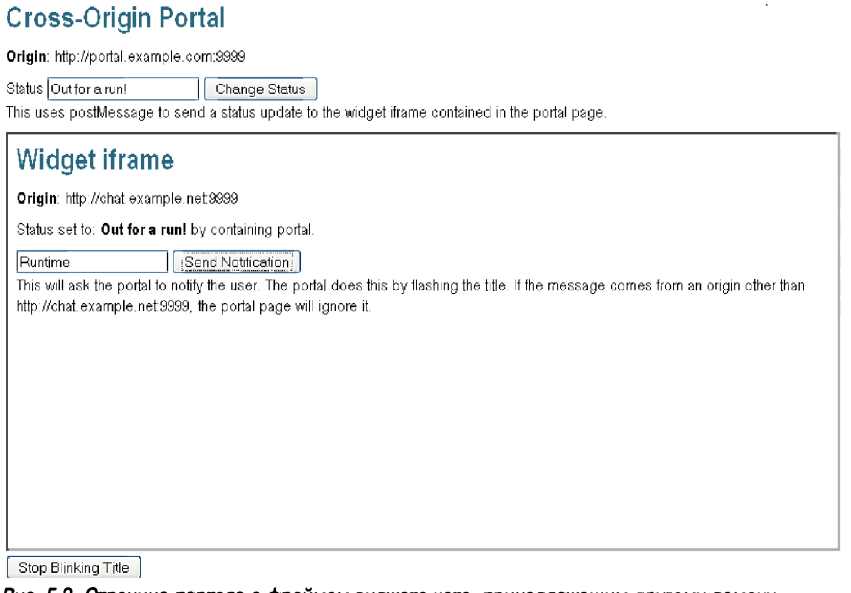


Рис. 5.3. Страница портала с фреймом виджета чата, принадлежащим другому домену

Примечание

В рассматриваемом примере приложения виджет чата не подключен ни к какой действующей системе обмена сообщениями по сети в режиме реального времени, и уведомлениями управляет пользователь приложения, щелкая на кнопке Отправить уведомление. В реальном приложении можно было бы использовать веб-сокеты, описанные в главе 6.

Для этого примера мы создали простые HTML-страницы `postMessagePortal.html` и `postMessageWidget.html`. Ниже описаны наиболее важные этапы создания страниц портала и виджета чата. Готовый код этого примера находится в папке `code/communication`.

Создание страницы портала

Прежде всего добавим фрейм виджета чата, принадлежащего другому источнику.

```
<iframe id="widget" src="http://chat.example.net:9999/
                         postMessageWidget.html"></iframe>
```

Затем добавим функцию `messageHandler()`, которая будет обрабатывать события сообщений, поступающих от виджета чата. Как видно из приведенного ниже кода, виджет запрашивает портал, чтобы тот отправил от его имени уведомление пользователю, инициировав мигание названия страницы. Чтобы удостовериться в том, что сообщение действительно было отправлено виджетом чата, выполняем проверку источника сообщения. Если сообщение поступило не от источника `http://chat.example.net:9999`, то страница портала просто игнорирует его.

```
var targetOrigin = "http://chat.example.net:9999";

function messageHandler(e) {
    if (e.origin == targetOrigin) {
        notify(e.data);
    } else {
        // Игнорировать сообщения из других источников
    }
}
```

Затем добавим функцию, обеспечивающую связь с виджетом чата. Эта функция использует метод `postMessage()` для отправки обновления статуса пользователя фрейму виджета на странице портала. В реальном приложении фрейму можно было бы передавать информацию о статусе пользователя (доступен, отсутствует и т.п.).

```
function sendString(s) {
    document.getElementById("widget").contentWindow.postMessage(s,
                                                        targetOrigin);
}
```

Создание страницы виджета чата

Прежде всего добавим функцию `messageHandler()`, которая будет обрабатывать события сообщений, поступающих от страницы портала. Как видно из приведенного ниже кода, виджет чата ожидает поступления сообщений об изменении статуса пользователя. Чтобы удостовериться в том, что сообщение действительно было отправлено страницей портала, выполняем проверку источника сообщения. Если сообщение поступило не от источника `http://portal.example.com:9999`, то виджет портала просто игнорирует его.

```
var targetOrigin = "http://portal.example.com:9999";

function messageHandler(e) {
    if (e.origin === "http://portal.example.com:9999") {
        document.getElementById("status").textContent = e.data;
    } else {
        // Игнорировать сообщения из других источников
    }
}
```

Затем добавим функцию, обеспечивающую связь со страницей портала. Виджет направляет порталу запрос, чтобы тот отправил от его имени уведомление пользователю, и использует метод `postMessage()` для отправки сообщения странице портала при получении нового сообщения чата, как показано в следующем примере.

```
function sendString(s) {
    window.top.postMessage(s, targetOrigin);
}
```

Окончательный код

В листинге 5.3 приведен завершенный код страницы портала `postMessagePortal.html`, а в листинге 5.4 — код страницы `postMessageWidget.html`.

Листинг 5.3. Содержимое файла `postMessagePortal.html`

```
<!DOCTYPE html>
<title>Портал [http://portal.example.com:9999]</title>
<link rel="stylesheet" href="styles.css">
<style>
    iframe {
```

```
        height: 400px;
        width: 800px;
    }
</style>
<link rel="icon" href="http://apress.com/favicon.ico">
<script>

var defaultTitle = "Портал [http://portal.example.com:9999]";
var notificationTimer = null;

var targetOrigin = "http://chat.example.net:9999";

function messageHandler(e) {
    if (e.origin == targetOrigin) {
        notify(e.data);
    } else {
        // Игнорировать сообщения из других источников
    }
}

function sendString(s) {
    document.getElementById("widget").contentWindow.postMessage(s,
                                                    targetOrigin);
}


function notify(message) {
    stopBlinking();
    blinkTitle(message, defaultTitle);
}

function stopBlinking() {
    if (notificationTimer !== null) {
        clearTimeout(notificationTimer);
    }
    document.title = defaultTitle;
}

function blinkTitle(m1, m2) {
    document.title = m1;
    notificationTimer = setTimeout(blinkTitle, 1000, m2, m1)
}

function sendStatus() {
var statusText = document.getElementById("statusText").value;
    sendString(statusText);
}

function loadDemo() {
    document.getElementById("sendButton").addEventListener("click",
                                                    sendStatus, true);
    document.getElementById("stopButton").addEventListener("click",
                                                stopBlinking, true);
    sendStatus();
}
window.addEventListener("load", loadDemo, true);
window.addEventListener("message", messageHandler, true);
```

```
</script>

<h1>Кроссдоменный портал</h1>
<p><b>Origin</b>: http://portal.example.com:9999</p>
Состояние <input type="text" id="statusText" value="Online">
<button id="sendButton">Изменить состояние</button>
<p>
Этот источник использует метод postMessage для отправки сообщения
об обновлении состояния фрейму виджета, размещенного на странице
портала.
</p>
<iframe id="widget" src="http://chat.example.net:9999/
                         postMessageWidget.html"></iframe>
<p>
    <button id="stopButton">Прекратить мигание названия</button>
</p>
```

Листинг 5.4. Содержимое файла `postMessageWidget.html`

```
<!DOCTYPE html>
<title>Виджет</title>
<link rel="stylesheet" href="styles.css">
<script>

var targetOrigin = "http://portal.example.com:9999";

function messageHandler(e) {
    if (e.origin === "http://portal.example.com:9999") {
        document.getElementById("status").textContent = e.data;
    } else {
        // Игнорировать сообщения из других источников
    }
}

function sendString(s) {
    window.top.postMessage(s, targetOrigin);
}

function loadDemo() {
   document.getElementById("actionButton").addEventListener("click",
      function() {
         var messageText =
                     document.getElementById("messageText").value;
         sendString(messageText);
      }, true);

}
window.addEventListener("load", loadDemo, true);
window.addEventListener("message", messageHandler, true);

</script>
<h1>Фрейм виджета</h1>
<p><b>Origin</b>: http://chat.example.net:9999</p>
<p>Состояние: <strong id="status"></strong> установлено содержащим
   порталом.<p>
<div>
    <input type="text" id="messageText" value="Уведомление
                                               от виджета.">
```

```
    <button id="actionButton">Отправить уведомление</button>
</div>

<p>
Этот источник обращается к порталу с запросом отправки
уведомления пользователю. Портал отвечает на запрос миганием
названия страницы. Сообщения, поступившие от источника,
отличного от http://chat.example.net:9999, будут
игнорироваться страницей портала.
</p>
```

Приложение в действии

Чтобы увидеть, как работает созданное приложение, необходимо соблюсти два условия: обе страницы должны загружаться с веб-сервера, причем из разных доменов. Если у вас есть доступ к нескольким серверам (например, к двум HTTP-серверам Apache) в разных доменах, можете разместить файлы по одному на двух различных серверах и запустить демонстрационный пример. Другой возможный способ состоит в том, чтобы использовать сервер SimpleHTTPServer среды разработки Python, развернутой на локальном компьютере в соответствии с описанной ниже процедурой.

1. Обновите содержимое файла `hosts` в Windows (`C:\Windows\system32\drivers\etc\hosts`) или Linux (`/etc/hosts`), добавив в него две записи, ссылающиеся на локальный хост (IP-адрес 127.0.0.1), как показано ниже.

   ```
   127.0.0.1  chat.example.net
   127.0.0.1  portal.example.com
   ```

Примечание

Чтобы новые DNS-записи гарантированно вступили в силу, следует перезапустить браузер после внесения изменений в файл `hosts`.

2. Установите среду разработки Python вместе с входящей в нее упрощенной версией веб-сервера SimpleHTTPServer.
3. Перейдите в каталог, в котором находятся оба файла примера (`postMessageParent.html` и `postMessageWidget.html`).
4. Запустите в этом каталоге среду Python из командной строки, используя следующие параметры:

   ```
   python -m SimpleHTTPServer 9999;
   ```

5. Откройте браузер и выполните переход по следующему адресу:

   ```
   http://portal.example.com:9999/postMessagePortal.html
   ```

На экране должна отобразиться страница, представленная на рис. 5.3.

XMLHttpRequest Level 2

XMLHttpRequest — это программный интерфейс, без которого реализация технологии AJAX была бы просто невозможной. Об XMLHttpRequest и AJAX написано немало книг.

XMLHttpRequest Level 2 — новая, значительно улучшенная версия XMLHttpRequest. В этой главе рассматриваются усовершенствования, введенные в XMLHttpRequest Level 2. Эти усовершенствования затронули следующие области:

- кроссдоменные XMLHttpRequest-запросы;
- события `progress`.

Кроссдоменные XMLHttpRequest-запросы

Раньше XMLHttpRequest-запросы подчинялись *политике ограничения домена* (same-origin policy), которая разрешает отправку запросов только в пределах домена текущей страницы. Спецификация XMLHttpRequest Level 2 позволяет выполнять кроссдоменные XMLHttpRequest-запросы с помощью протокола Cross Origin Resource Sharing (CORS), в котором используется понятие *источника* (origin), обсуждавшееся нами в одном из предыдущих разделов.

Кроссдоменные HTTP-запросы содержат заголовок `header`. С помощью этого заголовка серверу передается информация об источнике запроса. Этот заголовок защищен браузером и не может быть изменен кодом приложения. По существу, это сетевой эквивалент свойства `origin` событий сообщений, используемых средством Cross Document Messaging. Заголовок `origin` отличается от старого заголовка `referer`[1] тем, что последний содержит полный URL-адрес, включая путь. Поскольку путь может содержать критически важную информацию, браузеры иногда не пересылают его, пытаясь обеспечить защиту пользовательских данных. В то же время браузеры всегда отправляют заголовок `origin`, если он необходим.

Кроссдоменные XMLHttpRequest-запросы позволяют создавать веб-приложения, которые используют службы, расположенные в разных источниках. Например, если вы хотите разместить веб-приложение, которое использует статическое содержимое из одного источника, а службы AJAX — из другого, то для организации обмена сообщениями между этими двумя источниками можно использовать кроссдоменные XMLHttpRequest-запросы. Если бы не возможности XMLHttpRequest, то сообщениями можно было бы обмениваться лишь в пределах одного источника. Это наложило бы некоторые ограничения на параметры развертывания приложений. Например, вы вынуждены были бы развернуть веб-приложение в том же домене или настроить подчиненный домен.

Как показано на рис. 5.4, кроссдоменные XMLHttpRequest-запросы позволяют агрегировать на клиентской стороне содержимое из различных источников. Кроме того, можно получать доступ к защищенному содержимому, используя механизм учетных записей пользователей, если целевой сервер это позволяет, предоставляя пользователям прямой доступ к персонализированным данным. В то же время агрегация содержимого на стороне сервера принудительно пропускает все содержимое через серверную инфраструктуру, тем самым создавая узкое место для трафика.

В случае критически важных операций — например, запроса, содержащего учетные данные пользователя, или запроса, использующего метод, отличный от методов GET или POST, — спецификация CORS требует, чтобы браузер отправлял на сервер предварительный запрос с методом OPTIONS для проверки того, что данное действие поддерживается и разрешено. Это означает, что для успешной передачи данных может потребоваться, чтобы сервер поддерживал спецификацию CORS. В листингах 5.5 и 5.6 представлены заголовки, участвующие в кроссдоменном обмене сообщениями между страницей на хосте `www.example.com` и службой на хосте `www.example.net`.

---

[1] Давно закравшуюся в спецификации грамматическую ошибку (правильное написание этого слова — *referrer*) было решено не исправлять из соображений совместимости. — *Примеч. ред.*

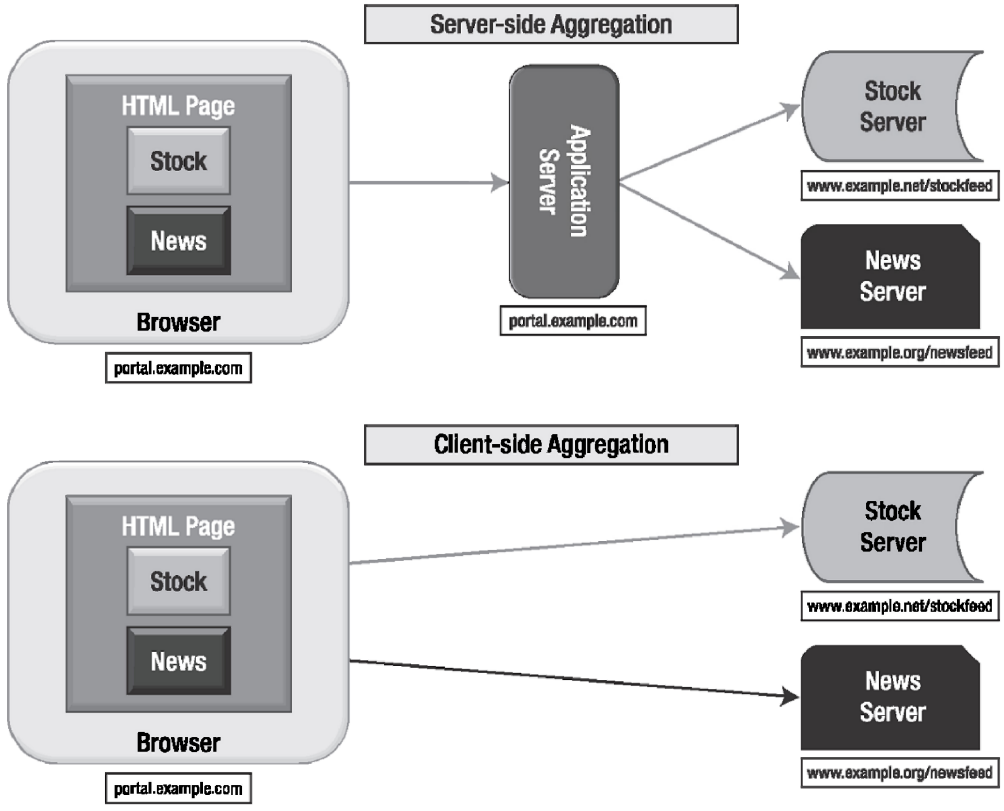


Рис. 5.4. Различия между агрегацией на стороне сервера и на стороне клиента

Листинг 5.5. Пример заголовков запроса

```
POST /main HTTP/1.1
Host: www.example.net
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.3)
 Gecko/20090910 Ubuntu/9.04 (jaunty) Shiretoko/3.5.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
        */*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/
Origin: http://www.example.com
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
```

Листинг 5.6. Пример заголовков ответа

```
HTTP/1.1 201 Created
Transfer-Encoding: chunked
```

```
Server: Kaazing Gateway
Date: Mon, 02 Nov 2009 06:55:08 GMT
Content-Type: text/plain
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Credentials: true
```

События состояния запроса

Одно из наиболее важных усовершенствований в спецификации XMLHttpRequest касается событий, указывающих на состояние запроса. В предыдущей версии спецификации XMLHttpRequest имелось только одно событие: `readystatechange`. Однако самое главное то, что это событие реализовано по-разному в различных браузерах. Так, состояние readyState 3 (интерактивный режим), соответствующее получению данных от сервера по частям, вообще никогда не генерируется в Internet Explorer. Кроме того, события `readystatechange` не предоставляли информацию о состоянии загрузки. Реализация индикаторов процесса загрузки данных была далеко не тривиальной задачей и не могла обойтись без участия серверной стороны.

Спецификация XMLHttpRequest Level 2 вводит события изменения состояния запроса с описательными именами. В табл. 5.2 приведены имена новых событий. Можно перехватывать любое из этих событий, задавая функцию обратного вызова в атрибуте обработчика событий. Например, если возникло событие `loadstart`, вызывается функция обратного вызова, указанная в свойстве `onloadstart`.

Таблица 5.2. Новые имена событий изменения состояния запроса в спецификации XMLHttpRequest Level 2

Имя события
`loadstart`
`progress`
`abort`
`error`
`load`
`loadend`

Прежнее свойство `readyState` и события `readystatechange` сохранены для обеспечения обратной совместимости.

Замечание по поводу атрибута `readyState`

В описании события `readystatechange` (сохранено для обеспечения обратной совместимости), приведенном в спецификации XMLHttpRequest Level 2, говорится о том, что в силу исторических причин моменты времени, в которые происходит изменение значения атрибута `readyState`, являются в некоторой степени произвольными.

Поддержка спецификации XMLHttpRequest Level 2 в браузерах

Как видно из табл. 5.3, на момент написания книги спецификация HTML5 XMLHttpRequest Level 2 уже поддерживалась многими браузерами.

Таблица 5.3. Поддержка спецификации HTML5 XMLHttpRequest браузерами

Браузер	Описание
Chrome	Поддерживается в версии 2.0 и выше
Firefox	Поддерживается в версии 3.5 и выше
Internet Explorer	Не поддерживается
Opera	Не поддерживается
Safari	Поддерживается в версии 4.0 и выше

Поскольку не все браузеры поддерживают данную спецификацию, целесообразно выполнять соответствующую проверку, прежде чем использовать описанные элементы. Об организации такой проверки программным путем говорится далее.

Программный интерфейс XMLHttpRequest

В этом разделе мы рассмотрим использование программного интерфейса XMLHttpRequest более подробно. В качестве примера создадим простую HTML-страницу `crossOriginUpload.html`. Используемый ниже код содержится в папке `code/communication`.

Проверка поддержки в браузере

Прежде чем пытаться использовать функциональность XMLHttpRequest Level 2, например кроссдоменные запросы, целесообразно предварительно убедиться в том, что она поддерживается браузером. Для этого, например, можно проверить, доступно ли в объекте `XMLHttpRequest` свойство `withCredentials`, как показано в листинге 5.7.

Листинг 5.7. Проверка доступности поддержки кроссдоменных запросов объектом `XMLHttpRequest`

```
var xhr = new XMLHttpRequest()
if (typeof xhr.withCredentials === undefined) {
    document.getElementById("support").innerHTML =
        "Ваш браузер <strong>не</strong> поддерживает кроссдоменные
         XMLHttpRequest-запросы";
} else {
    document.getElementById("support").innerHTML =
        "Ваш браузер <strong>поддерживает</strong>кроссдоменные
         XMLHttpRequest-запросы";
}
```

Выполнение кроссдоменных запросов

Чтобы выполнить кроссдоменный запрос, сначала необходимо создать новый объект `XMLHttpRequest`, как показано ниже.

```
var crossOriginRequest = new XMLHttpRequest()
```

После этого можно выполнить запрос, указав адрес другого источника.

```
crossOriginRequest.open("GET", "http://www.example.net/stockfeed",
                        true);
```

Следует обязательно обрабатывать возможные ошибки. Существует множество причин, по которым запрос может не быть выполненным успешно. К этому, например, могут привести сбои в сети, отказ в доступе или отсутствие поддержки протокола CORS на целевом сервере.

События, изменяющие состояние запроса

Вместо пронумерованных состояний, представляющих различные этапы выполнения запроса или получения ответа, спецификация XMLHttpRequest Level 2 предоставляет именованные события `progress`. Любое из этих событий можно перехватывать, указав функцию обратного вызова в атрибуте обработчика событий.

Использование функции обратного вызова для обработки событий `progress` иллюстрирует листинг 5.8. В этих событиях имеются поля, в которых указывается суммарный объем передаваемых данных и объем уже переданных данных, а также булево значение, показывающее, известен ли суммарный объем данных (этот объем может быть не известен в случае потокового трафика HTTP).

С такими же полями отправляет события и запрос `XMLHttpRequest.upload`.

Листинг 5.8. Использование события `onprogress`

```
crossOriginRequest.onprogress = function(e) {
    var total = e.total;
    var loaded = e.loaded;

    if (e.lengthComputable) {
        // Выполнить действия, зависящие от состояния запроса
    }
}


crossOriginRequest.upload.onprogress = function(e) {
    var total = e.total;
    var loaded = e.loaded;

    if (e.lengthComputable) {
        // Выполнить действия, зависящие от состояния запроса
    }
}
```

Создание приложения, использующего XMLHttpRequest-запросы

В этом примере мы рассмотрим выгрузку геолокационных координат на вебсервер, принадлежащий другому домену. Новые события `progress` мы используем для наблюдения за состоянием HTTP-запроса, в том числе за относительной долей переданных данных. Результаты работы приложения представлены на рис. 5.5.

В этом примере используется файл `crossOriginUpload.html`. Опишем наиболее важные этапы создания страницы, представленной на рис. 5.5. Соответствующий код находится в папке `code/communication`.

Прежде всего создадим новый объект `XMLHttpRequest`.

```
var xhr = new XMLHttpRequest();
```

Затем проверим, поддерживает ли браузер функциональность XMLHttpRequest.

```
if (typeof xhr.withCredentials === undefined) {
  document.getElementById("support").innerHTML =
           "Ваш браузер <strong>не</strong> поддерживает
            кроссдоменные XMLHttpRequest-запросы";
} else {
    document.getElementById("support").innerHTML =
```

```
                "Ваш браузер <strong>поддерживает</strong>
                 кроссдоменные XMLHttpRequest-запросы";
}
```

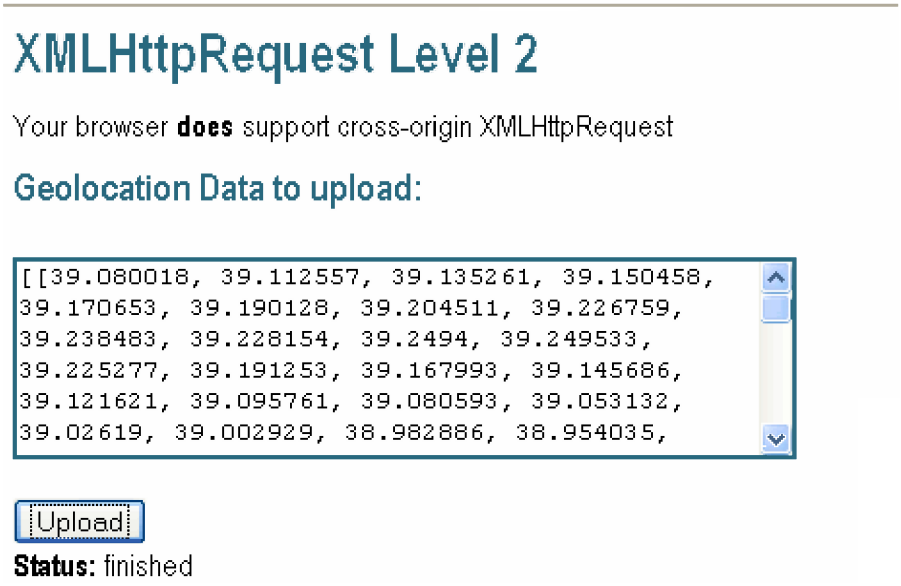


Рис. 5.5. Веб-приложение, выгружающее геолокационные данные на веб-сервер

Далее зададим функции обратного вызова для обработки событий progre и вычисления относительных долей выгружаемых и загруженных данных.

```
xhr.upload.onprogress = function(e) {
  var ratio = e.loaded / e.total;
  setProgress(ratio + "% uploaded");
}
xhr.onprogress = function(e) {
  var ratio = e.loaded / e.total;
  setProgress(ratio + "% downloaded");
}
xhr.onload = function(e) {
  setProgress("finished");
}
xhr.onerror = function(e) {
  setProgress("error");
}
```

Наконец, откроем запрос и отправим строку, содержащую закодированные г локационные данные. Это будет кроссдоменный запрос, поскольку целевое мес положение задано с помощью URL-адреса, который указывает на источник, отл ный от источника самой HTML-страницы.

```
var targetLocation = "http://geodata.example.net:9999/upload";
xhr.open("POST", targetLocation, true);

geoDataString = dataElement.textContent;
xhr.send(geoDataString);
```

Окончательный код

В листинге 5.9 приведен код приложения, содержащийся в файле crossOriginalUpload.html.

Листинг 5.9. Содержимое файла crossOriginalUpload.html

```
<!DOCTYPE html>
<title>Выгрузка геолокационных данных</title>
<link rel="stylesheet" href="styles.css">
<link rel="icon" href="http://apress.com/favicon.ico">
<script>
function loadDemo() {
    var dataElement = document.getElementById("geodata");
    dataElement.textContent =
                JSON.stringify(geoData).replace(",", ", ", "g");

    var xhr = new XMLHttpRequest()
    if (typeof xhr.withCredentials === undefined) {
        document.getElementById("support").innerHTML =
            "Ваш браузер <strong>не</strong> поддерживает
             кроссдоменные XMLHttpRequest-запросы";
    } else {
        document.getElementById("support").innerHTML =
            "Ваш браузер <strong>поддерживает</strong>
             кроссдоменные XMLHttpRequest-запросы";
    }

    var targetLocation = "http://geodata.example.net:9999/upload";

    function setProgress(s) {
        document.getElementById("progress").innerHTML = s;
    }

    document.getElementById("sendButton").addEventListener("click",
        function() {
            xhr.upload.onprogress = function(e) {
                var ratio = e.loaded / e.total;
                setProgress(ratio + "% uploaded");
            }

            xhr.onprogress = function(e) {
              var ratio = e.loaded / e.total;
              setProgress(ratio + "% downloaded");
            }

            xhr.onload = function(e) {
                setProgress("finished");
            }

            xhr.onerror = function(e) {
                setProgress("error");
            }

            xhr.open("POST", targetLocation, true);

            geoDataString = dataElement.textContent;
            xhr.send(geoDataString);
        }, true);
```

```
}
window.addEventListener("load", loadDemo, true);

</script>
<h1>XMLHttpRequest Level 2</h1>
<p id="support"></p>

<h4>Выгружаемые геолокационные данные:</h4>
<textarea id="geodata">
</textarea>
</div>

<button id="sendButton">Выгрузить</button>

<script>
geoData = [[39.080018000000003, 39.112557000000002, 39.135261,
39.150458, 39.170653000000001, 39.190128000000001,
39.204510999999997, 39.226759000000001, 39.238483000000002,
39.228154000000004, 39.249400000000001, 39.249533,
39.225276999999998, 39.191253000000003, 39.167993000000003,
39.145685999999998, 39.121620999999998, 39.095761000000003,
39.080593, 39.053131999999998, 39.02619, 39.002929000000002,
38.982886000000001, 38.954034999999998, 38.944926000000002,
38.919960000000003, 38.925261999999996, 38.934922999999998,
38.949373000000001, 38.950133999999998, 38.952649000000001,
38.969692000000002, 38.988512999999998, 39.010652,
39.033088999999997, 39.053493000000003, 39.072752999999999],
[-120.15724399999999, -120.15818299999999, -120.15600400000001,
-120.14564599999999, -120.141285, -120.10889900000001,
-120.09528500000002, -120.077596, -120.045428, -120.0119,
-119.98897100000002, -119.95124099999998, -119.93270099999998,
-119.927131, -119.92685999999999, -119.92636200000001,
-119.92844600000001, -119.911036, -119.942834, -119.94413000000002,
-119.94555200000001, -119.95411000000001, -119.941327,
-119.94605900000001, -119.97527599999999, -119.99445, -120.028998,
-120.066335, -120.07867300000001, -120.089985, -120.112227,
-120.09790700000001, -120.10881000000001, -120.116692, -120.117847,
-120.11727899999998, -120.143981999999999]];
</script>

<p>
    <b>Состояние: </b> <span id="progress">ready</span>
</p>
```

Приложение в действии

Чтобы увидеть, как работает созданное приложение, необходимо соблюсти два условия: страницы должны загружаться из разных доменов, а целевая страница должна загружаться с веб-сервера, распознающего заголовки CORS. Удовлетворяющий требованиям спецификации CORS сценарий на языке Python, способный обрабатывать поступающие кроссдоменные XMLHttpRequest-запросы, включен в код примеров для этой главы. Можете использовать демонстрационный код примера на своем локальном компьютере, выполнив следующие действия.

1. Обновите содержимое файла `hosts` в Windows (`C:\Windows\system32\drivers\etc\hosts`) или Linux (`/etc/hosts`), добавив в него две записи, ссылающиеся на локальный хост (IP-адрес 127.0.0.1), как показано ниже.

```
127.0.0.1  geodata.example.net
127.0.0.1  portal.example.com
```

Примечание

Чтобы новые DNS-записи гарантированно вступили в силу, следует перезапустить браузер после внесения изменений в файл `hosts`.

2. Установите среду разработки Python вместе с входящей в нее упрощенной версией веб-сервера SimpleHTTPServer, если это не было сделано ранее при выполнении предыдущего примера приложения.
3. Перейдите в каталог, в котором находятся файлы примера (`crossOriginUpload.html`) и CORS-совместимого серверного сценария на языке Python (`CORSServer.py`).
4. Запустите в этом каталоге среду Python, выполнив следующую команду:
   `python CORSServer.py 9999;`
5. Откройте браузер и выполните переход по следующему адресу:
   `http://portal.example.com:9999/crossOriginUpload.html`

На экране должна отобразиться страница, представленная на рис. 5.5.

Дополнительные рекомендации

Некоторые методики, не нашедшие отражения в основных примерах, приведенных в данной книге, могут быть весьма полезными во многих типах HTML5-приложений. Ниже представлены небольшие фрагменты кода, представляющие практический интерес, вместе с соответствующими комментариями и рекомендациями.

Структурированные данные

В ранних версиях метода `postMessage()` поддерживались только строки. Более поздние редакции допускают другие типы данных, включая объекты JavaScript, объекты `imageData` и файлы. Поддержка различных типов данных браузерами будет меняться по мере разработки спецификации.

В некоторых браузерах к объектам JavaScript, которые могут пересылаться с помощью метода `postMessage()`, применяются те же ограничения, что и к данным в формате JSON. В частности, циклические структуры данных могут оказаться неприемлемыми. Примером таких данных может служить список, одним из элементов которого является ссылка на самого себя.

Подавление фреймов

Подавление фреймов (framebusting) — это методика, позволяющая убедиться в том, что содержимое не загружено в элемент `iframe`. Приложение может обнаружить, что его окно не является верхним (`window.top`) в иерархии фреймов, и выйти за пределы содержащего его фрейма, как показано в приведенном ниже примере.

```
if (window !== window.top) {
    window.top.location = location;
}
```

Однако вполне может случиться так, что для некоторых страниц вы хотели бы разрешить помещение своего содержимого во фрейм. Одно из возможных решений состоит в том, чтобы использовать метод `postMessage()` для осуществления

"согласования" между кооперирующимися фреймами и содержащими их страницами, как показано в листинге 5.10.

Листинг 5.10. Использование метода `postMessage()` во фрейме для согласования с доверенной страницей

```
var framebustTimer;
var timeout = 3000; // трехсекундный таймаут

if (window !== window.top) {
    framebustTimer = setTimeout(
        function() {
            window.top.location = location;
        }, timeout);
}

window.addEventListener("message", function(e) {
    switch(e.origin) {
        case trustedFramer:
            clearTimeout(framebustTimer);
            break;
    }
}, true);
```

Резюме

В этой главе было продемонстрировано, каким образом возможности, обеспечиваемые спецификациями HTML5 Cross Document Messaging и XMLHttpRequest Level 2, могут быть использованы для создания профессиональных приложений, позволяющих осуществлять безопасный обмен данными между разными источниками.

Сначала мы обсудили метод `postMessage()` и концепцию безопасности источника (два ключевых элемента, связанных с обменом сообщениями в HTML5), а затем показали, как с помощью метода `postMessage()` обеспечить обмен сообщениями между фреймами, вкладками и окнами.

Мы также обсудили спецификацию XMLHttpRequest Level 2 — улучшенную версию спецификации XMLHttpRequest. Было показано, в чем именно состоят улучшения, наиболее важные из которых связаны с событиями `readystatechange`. Далее было продемонстрировано, как использовать возможности программного интерфейса XMLHttpRequest для отправки запросов между различными источниками и как использовать новые события `progress`.

Наконец, мы завершили главу несколькими полезными практическими примерами. В следующей главе демонстрируется использование веб-сокетов HTML5, обеспечивающих поразительно простой способ отправки данных приложению при минимальном служебном сетевом трафике.

Глава 6

Веб-сокеты

В этой главе изучаются возможности самого мощного коммуникационного средства HTML5 — спецификации WebSocket, которая реализует полностью двунаправленный канал связи с Интернетом через единственный веб-сокет. WebSocket — не просто очередное инкрементное улучшение стандартных коммуникационных механизмов HTML. Эта спецификация знаменует собой значительный прогресс, особенно в области разработки веб-приложений, управляемых событиями в режиме реального времени.

Протокол WebSocket является настолько значительным усовершенствованием по сравнению с запутанными приемами, использовавшимися прежде для имитации дуплексных соединений в браузерах, что это побудило Яна Хиксона, сотрудника Google и ведущего разработчика спецификации HTML5, сделать следующее замечание.

> *"Уменьшение объемов передаваемых данных, исчисляемых килобайтами, до 2 байтов... и уменьшение времени ожидания со 150 мс до 50 мс нельзя считать мелочью. В действительности уже одни только эти два фактора заставляют компанию Google всерьез заинтересоваться технологией WebSocket."*
>
> — `www.ietf.org/mail-archive/web/hybi/current/msg00784.html`

Мы подробно расскажем, благодаря каким факторам веб-сокеты HTML5 обеспечивают столь разительное улучшение, и вы сами убедитесь в том, что спецификация HTML5 WebSocket в один миг переводит в категорию устаревших такие прежние решения на основе технологий Comet и AJAX, как опрос, продленный опрос и стриминг.

Веб-сокеты в HTML5

Проанализируем, за счет чего веб-сокетам HTML5 удается снизить чрезмерный служебный сетевой трафик и уменьшить время задержки, сравнив для этого HTTP-решения, обеспечивающие двухстороннюю связь между сервером и браузером в режиме реального времени, с возможностями технологии WebSocket.

Работа в реальном времени и HTTP

Обычно, когда пользователь посещает веб-страницу с помощью браузера, веб-серверу, на котором размещена эта страница, направляется HTTP-запрос. Веб-сервер подтверждает получение запроса и посылает браузеру ответ. Однако во многих случаях — например, если речь идет о биржевых котировках, новостях, продаже билетов, расписаниях движения транспорта или считывании показаний медицинских приборов, — к тому времени, когда браузер обновит страницу, сведения, поступившие вместе с ответом, могут оказаться устаревшими. Если вы хотите получать самую свежую информацию в режиме реального времени, можно постоянно обновлять страницу вручную, но совершенно очевидно, что такое решение не является оптимальным.

В настоящее время все попытки создания быстрых веб-приложений так или иначе сводятся к использованию технологии опроса и других разновидностей сер-

верных push-технологий, самой примечательной из которых является технология Comet, в которой завершение HTTP-ответа задерживается сервером для доставки данных клиенту.

При использовании метода *опроса* (polling) браузер регулярно отправляет серверу запросы через определенные промежутки времени и получает на них немедленные ответы. Этот метод был первой попыткой обеспечить доставку информации браузерам в режиме реального времени. Очевидно, что это решение хорошо подходит для ситуаций, в которых периодичность появления сообщений, подлежащих отправке браузеру, точно известна, поскольку это позволяет синхронизировать клиентские запросы и отправлять их лишь тогда, когда на сервере имеется новая информация. Однако в реальных условиях часто невозможно предсказать точно, когда именно следует ожидать появления новых данных, и если это происходит довольно редко, то соединение с сервером будет понапрасну открываться и закрываться множество раз, и подавляющая часть запросов окажется бесполезной.

При использовании метода *продленного опроса* (long-polling) браузер направляет серверу запрос, и тот удерживает запрос открытым в течение указанного периода времени. Если в это время сервер получает уведомление, он отправляет клиенту ответ, содержащий сообщение. В противном случае сервер отправляет ответ, закрывающий запрос. Следует, однако, понимать, что в случае интенсивного обмена небольшими порциями данных метод продленного опроса не обеспечивает никаких существенных преимуществ по сравнению с обычным методом опроса.

При использовании метода *стриминга* (streaming) браузер отправляет завершенный запрос, но сервер отправляет ответ, который непрерывно обновляется и остается открытым в течение сколь угодно длительного (или указанного) промежутка времени. Как только появляется сообщение, готовое к отправке, ответ обновляется, но сигнал о завершении ответа не посылается, и соединение остается открытым для доставки будущих сообщений. В результате этого клиент получает новые сообщения практически без задержки. Вместе с тем, поскольку стриминг остается инкапсулированным в HTTP, оказавшиеся на пути сообщений брандмауэры и прокси-серверы, настроенные для буферизации данных, могут увеличивать время доставки сообщений. Поэтому при обнаружении прокси-сервера, использующего буферизацию, многие решения на основе стриминга начинают работать в режиме продленного опроса. Помочь избежать буферизации ответов может также использование SSL/TLS-соединений, но в этом случае необходимость установления и разрыва каждого соединения оборачивается дополнительной нагрузкой для сервера.

В конечном счете каждый из вышеперечисленных методов предоставления информации в реальном времени использует HTTP-заголовки запросов и ответов, в которых содержится много лишних дополнительных заголовочных данных, приводящих к увеличению времени задержки. Но важнее всего то, что для обеспечения полностью дуплексной связи требуется нечто большее, нежели просто соединение для передачи данных от сервера к клиенту. Пытаясь имитировать полностью дуплексные соединения поверх полудуплексного протокола HTTP, многие современные решения используют два соединения: одно для передачи данных от клиента к серверу, а второе — от сервера к клиенту. Необходимость обслуживания этих соединений и обеспечения их согласованной работы влечет за собой дополнительные накладные расходы в виде возрастания нагрузки на сервер и усложняет организацию всего процесса. Попросту говоря, протокол HTTP не предназначался для обеспечения дуплексной связи в реальном времени, как это видно из рис. 6.1, иллюстрирующего все сложности, связанные с созданием веб-приложения, которое в реальном времени отображает поступающие из соответствующего источника данные, используя модель "публикация/подписка" поверх полудуплексного протокола HTTP.

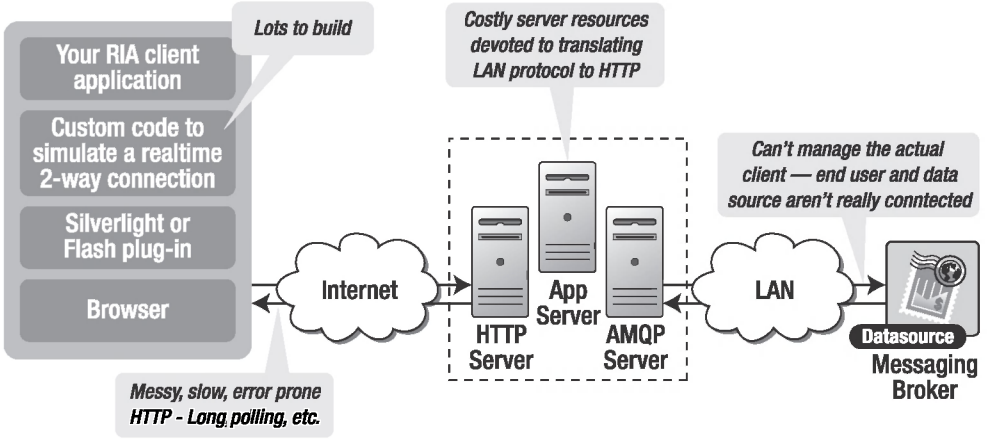


Рис. 6.1. Сложность создания приложений, работающих в реальном времени по протоколу HTTP

При попытках масштабирования этих решений ситуация еще более усугубляется. Имитация двухсторонней связи с браузером поверх HTTP увеличивает риск сбоев и вносит ряд сложностей, затрудняющих масштабирование. Даже если конечным пользователям и могут нравиться веб-приложения, обменивающиеся данными с сервером настолько быстро, что создается впечатление работы в режиме реального времени, эта "скорость" дается высокой ценой. За нее приходится платить дополнительным временем задержки, добавочным сетевым трафиком и возрастанием нагрузки на процессор.

Концепция веб-сокетов в HTML5

Первоначально веб-сокеты HTML5 были определены Яном Хиксоном (ведущий разработчик спецификации HTML5) как объекты "TCPConnection" в разделе "Communications" спецификации HTML5. Протокол веб-сокетов эволюционировал и в настоящее время выделен в самостоятельную спецификацию WebSocket (существующую наравне с такими спецификациями, как Geolocation, Web Workers и т.п.), чтобы ее обсуждение носило более целенаправленный характер.

Что может быть общего между веб-сокетами и моделями поездов?

Говорит Питер: "Ян Хиксон — большой любитель железнодорожных моделей. Разработкой компьютерных систем управления движением железнодорожных поездов он увлекся еще в 1984 году, когда компания Marklin впервые применила в своих моделях цифровой контроллер.

Как раз тогда, когда Ян добавлял в спецификацию HTML5 определение TCPConnection, он разрабатывал программу для управления моделью поезда из браузера, организуя связь между браузером и поездом на основе "висячих GET-запросов" ("hanging GET") и XHR. Создать такую программу было бы намного легче, если бы связь с браузером можно было поддерживать через сокеты, в значительной степени аналогично тому, как это делается в традиционной модели асинхронной связи "клиент-сервер", применяемой настольными приложениями. Именно благодаря этой вдохновляющей идее о том, что *могло бы* быть, и закрутились колеса (поезда), и поезд WebSocket отправился со станции. Следующая остановка — веб-сайты, работающие в режиме реального времени."

Установка соединения по протоколу WebSocket

Для установления WebSocket-соединения клиент и сервер должны расширить протокол от HTTP до WebSocket в процессе обмена начальными подтверждениями (*квитирование*, или "рукопожатие"), как показано в листинге 6.1. Заметим, что приведенное описание соответствует черновому варианту 76 протокола; в будущих вариантах спецификации процедура квитирования соединения может измениться.

Листинг 6.1. Квитирование перехода к протоколу WebSocket в сеансе связи

```
От клиента серверу:
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 4@1 46546xW%0l 1 5
Origin: http://example.com

[8-байтовый ключ защиты]

От сервера клиенту:
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://example.com
WebSocket-Location: ws://example.com/demo
WebSocket-Protocol: sample

[16-байтовый хеш-код ответа]
```

Как только соединение установлено, WebSocket-сообщения могут пересылаться в обоих направлениях между клиентом и сервером в полностью дуплексном режиме. Это означает, что обмен текстовыми сообщениями может осуществляться в дуплексном режиме, т.е. одновременно в обоих направлениях. В сети каждое сообщение начинается байтом `0x00` и заканчивается байтом `0xFF`, между которыми содержатся данные в формате UTF-8.

Интерфейс `WebSocket`

Наряду с определением протокола WebSocket в спецификации также содержится определение интерфейса `WebSocket`, предназначенного для использования JavaScript-приложениями (листинг 6.2).

Листинг 6.2. Интерфейс `WebSocket`

```
[Constructor(in DOMString url, in optional DOMString protocol)]
interface WebSocket {
  readonly attribute DOMString URL;

  // состояние готовности
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // работа в сети
  attribute Function onopen;
```

```
  attribute Function onmessage;
  attribute Function onclose;
  boolean send(in DOMString data);
  void close();
};
WebSocket implements EventTarget;
```

Использование интерфейса `WebSocket` не вызывает сложностей. Для установления соединения с удаленным хостом достаточно создать экземпляр объекта `WebSocket` и передать ему URL, представляющий оконечный узел, с которым необходимо связаться. Также заметьте, что для указания обычного и безопасного WebSocket-соединения используются префиксы `ws://` и `wss://` соответственно.

Соединение WebSocket устанавливается путем расширения протокола HTTP до протокола WebSocket в процессе первоначального "рукопожатия" между клиентом и сервером через одно и то же базовое соединение TCP/IP. Как только соединение установлено, кадры данных могут пересылаться в обоих направлениях между клиентом и сервером в полностью дуплексном режиме. Доступ к самому соединению осуществляется посредством события `message` и метода `send`, определенных интерфейсом `WebSocket`. Для обработки каждой фазы жизненного цикла соединения вы должны использовать обработчики асинхронных событий.

```
myWebSocket.onopen = function(evt)
 { alert("Соединение установлено ..."); };
myWebSocket.onmessage = function(evt)
 { alert( "Получено сообщение:  "  +  evt.data); };
myWebSocket.onclose = function(evt)
 { alert("Соединение закрыто."); };
```

Радикальное уменьшение служебного сетевого трафика и времени задержки

Итак, благодаря чему веб-сокеты оказываются столь эффективными? Для анализа этого вопроса проведем параллели между приложением на основе технологии опроса и приложением на основе веб-сокетов. Чтобы проиллюстрировать, как работает метод опроса, рассмотрим простое веб-приложение, в котором веб-страница запрашивает у брокера сообщений RabbitMQ биржевые данные в режиме реального времени, используя традиционную модель "публикация-подписка". Эти данные получаются путем опроса Java-сервлета, установленного на веб-сервере. Брокер сообщений RabbitMQ получает данные из фиктивного канала биржевых котировок, в котором котировки непрерывно обновляются. Веб-страница подключается к определенному биржевому каналу (теме в брокере сообщений), подписывается на него и ежесекундно опрашивает канал, выясняя наличие обновлений с помощью запросов XMLHttpRequest. В случае получения обновлений выполняются определенные вычисления, и биржевые данные отображаются в виде таблицы, как показано на рис. 6.2.

Все это выглядит заманчиво, но более глубокий анализ позволяет обнаружить, что с работой этого приложения связаны серьезные проблемы. Например, если воспользоваться консолью Firebug браузера Firefox, то можно будет увидеть, как буквально ежесекундно GET-запросы "бомбардируют" сервер. Активизировав расширение Просмотр HTTP-заголовков (Live HTTP Headers), вы обнаружите, что с каждым запросом связана передача непомерно большого объема данных заголовка. Об этом можно судить по листингам 6.3 и 6.4, в которых показаны заголовочные данные, связанные всего лишь с единственной парой "запрос-ответ".

COMPANY	SYMBOL	PRICE	CHANGE	SPARKLINE	OPEN	LOW	HIGH
THE WALT DISNEY COMPANY	DIS	27.65	0.56		27.09	24.39	29.80
GARMIN LTD.	GRMN	35.14	0.35		34.79	31.31	38.27
SANDISK CORPORATION	SNDK	20.11	-0.13		20.24	18.22	22.26
GOODRICH CORPORATION	GR	49.99	-2.35		52.34	47.11	57.57
NVIDIA CORPORATION	NVDA	13.92	0.07		13.85	12.47	15.23
CHEVRON CORPORATION	CVX	67.77	-0.53		68.30	61.49	75.11
THE ALLSTATE CORPORATION	ALL	30.88	-0.14		31.02	27.92	34.12
EXXON MOBIL CORPORATION	XOM	65.66	-0.86		66.52	59.87	73.17
METLIFE INC.	MET	35.58	-0.15		35.73	32.16	39.30

Рис. 6.2. JavaScript-приложение, доставляющее биржевые котировки

Листинг 6.3. Заголовок HTTP-запроса

```
GET /PollingStock//PollingStock HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
 rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
q=0.8
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/PollingStock/
Cookie: showInheritedConstant=false;
 showInheritedProtectedConstant=false;
 showInheritedProperty=false;
 showInheritedProtectedProperty=false;
 showInheritedMethod=false; showInheritedProtectedMethod=false;
 showInheritedEvent=false; showInheritedStyle=false;
 showInheritedEffect=false
```

Листинг 6.4. Заголовок HTTP-ответа

```
HTTP/1.x 200 OK
X-Powered-By: Servlet/2.5
Server: Sun Java System Application Server 9.1_02
Content-Type: text/html;charset=UTF-8
Content-Length: 21
Date: Sat, 07 Nov 2009 00:32:46 GMT
```

Ради интереса подсчитаем общее количество всех символов. Суммарный объем служебной информации в приведенных выше HTTP-запросе и ответе составляет 871 байт. Разумеется, это всего лишь пример, и в некоторых случаях эта величина может оказаться меньшей, но нередки также случаи, когда объем служебных данных превышает 2000 байт. В нашем примере приложения набор данных типичного биржевого сообщения по одной позиции содержит около 20 символов. Как нетрудно заметить, эти данные практически тонут в избыточном объеме служебной информации, которая не является для нас первостепенной.

А что произойдет, если попытаться развернуть это приложение для большого количества пользователей? Попробуем оценить объем балластной заголовочной информации, которая содержится в HTTP-запросе и ответе, используемых в нашем приложении, для трех различных случаев.

- **Случай A:** ежесекундный опрос со стороны 1000 клиентов. Сетевой трафик составляет (871 × 1000) = 871000 байт = 6968000 бит/с (6,6 Мбит/с).
- **Случай B:** ежесекундный опрос со стороны 10000 клиентов. Сетевой трафик составляет (871 × 10000) = 8710000 байт = 69680000 бит/с (66 Мбит/с).
- **Случай C:** ежесекундный опрос со стороны 100000 клиентов. Сетевой трафик составляет (871 × 100000) = 87100000 байт = 696800000 бит/с (665 Мбит/с).

Таким образом, по сети передается огромный объем лишней служебной информации. Посмотрим, что произойдет, если мы перестроим приложение для использования веб-сокетов HTML5, добавив на веб-страницу обработчик событий для получения сообщений с обновленными биржевыми котировками, поступающими от брокера сообщений (более подробно об этом будет говориться немного позже). Каждое из этих сообщений представляет собой кадр WebSocket, содержащий всего лишь два байта протокольных данных (вместо 871). Взгляните, как это влияет на объемы служебной информации, передаваемой по сети в каждом из трех случаев.

- **Случай A:** 1000 клиентов получают по одному сообщению в секунду. Сетевой трафик составляет (2 × 1000) = 2000 байт = 16000 бит/с (0,015 Мбит/с).
- **Случай B:** 10000 клиентов получают по одному сообщению в секунду. Сетевой трафик составляет (2 × 10000) = 20000 байт = 160000 бит/с (0,153 Мбит/с).
- **Случай C:** 100000 клиентов получают по одному сообщению в секунду. Сетевой трафик составляет (2 × 100000) = 200000 байт = 1600000 бит/с (1,526 Мбит/с).

Как видно из рис. 6.3, протокол HTML5 WebSocket позволяет резко снизить объем лишнего сетевого трафика по сравнению с решением, основанным на методе опроса.

А что можно сказать об уменьшении времени задержки? Взгляните на рис. 6.4. В его верхней части показаны величины времени задержки для решения, основанного на полудуплексном опросе. Если в этом примере предположить, что для прохождения пути от сервера к браузеру сообщению требуется 50 миллисекунд, то приложение, использующее метод опроса, вводит большую дополнительную задержку, поскольку после завершения ответа на сервер должен быть отправлен новый запрос. Для каждого нового запроса потребуется дополнительные 50 миллисекунд, в течение которых сервер не сможет отправлять сообщения браузеру, что приводит к дополнительному расходованию памяти сервера.

Нижняя часть рисунка иллюстрирует уменьшение времени задержки, обеспечиваемое решением на основе протокола WebSocket. Как только соединение расширено до WebSocket, сообщения могут "перетекать" от сервера к браузеру сразу же после их поступления. Для прохождения сообщения от сервера к браузеру попрежнему требуется 50 миллисекунд, но WebSocket-соединение все время остается открытым, и поэтому необходимость в отправке дополнительных запросов на сервер отпадает.

Спецификация HTML5 WebSocket знаменует огромный прогресс в отношении масштабируемости интернет-приложений, работающих в реальном времени. Как уже было показано, веб-сокеты HTML5 позволяют уменьшить объем служебного трафика, связанного с передачей HTTP-заголовков, в 500, а в некоторых случаях — в зависимости от размера HTTP-заголовков — даже в 1000 раз, а время запаздывания — в 3 раза.

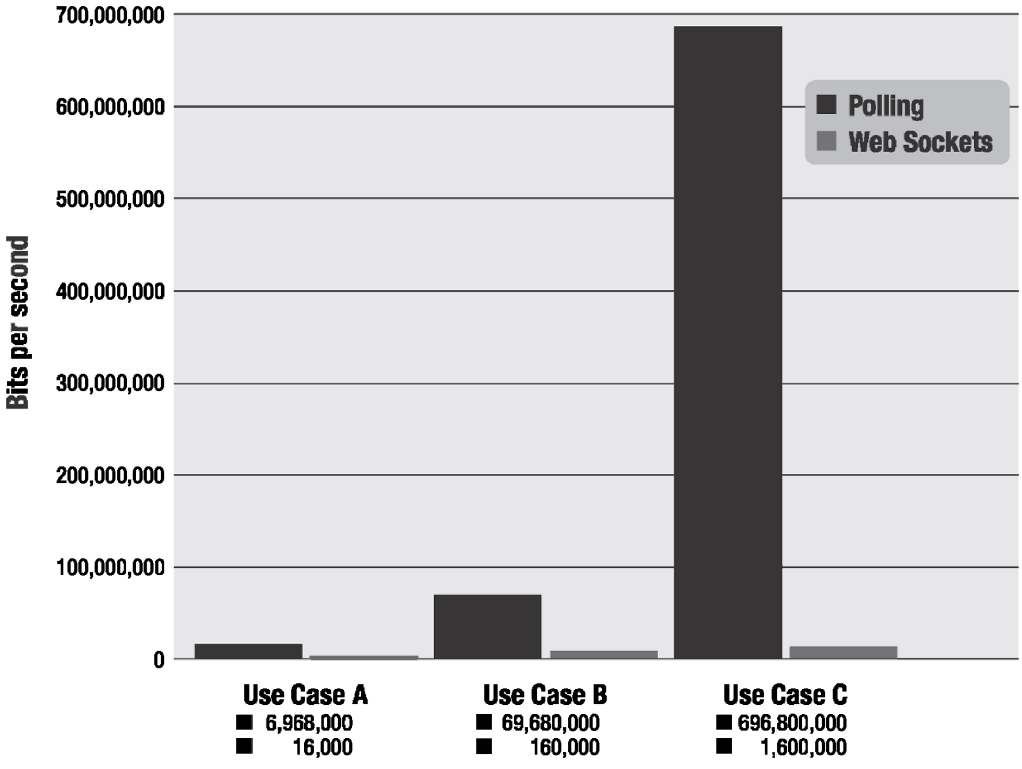


Рис. 6.3. Сравнение объемов служебного сетевого трафика для вариантов приложения, использующих технологии опроса и WebSocket

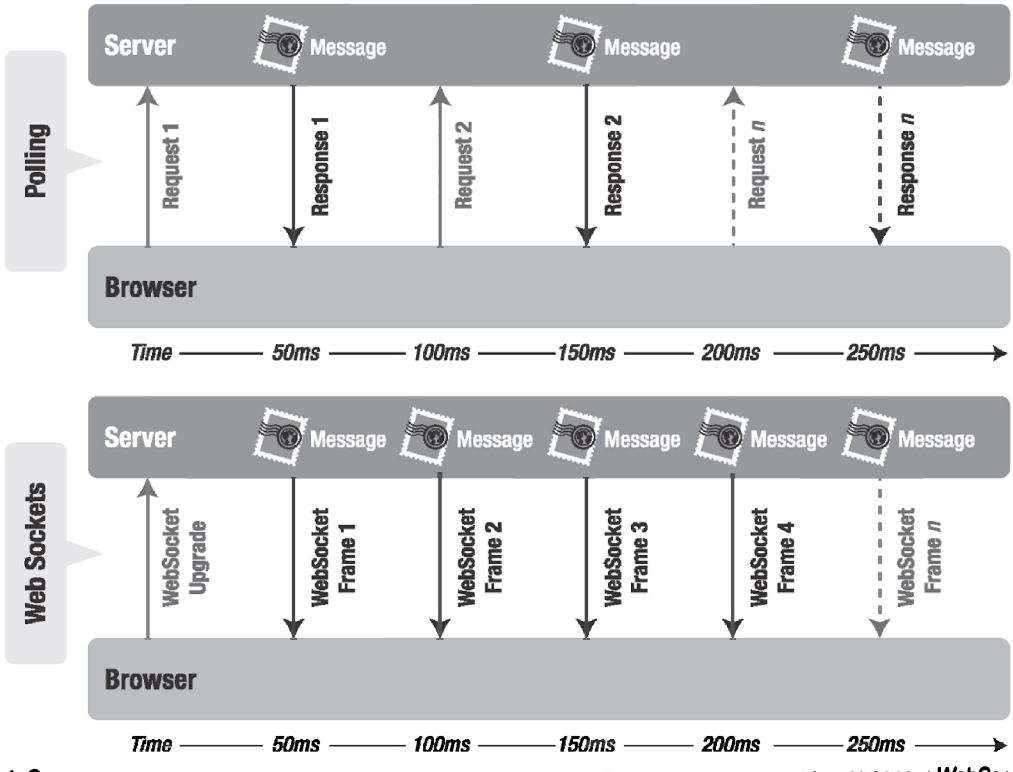


Рис. 6.4. Сравнение времени задержки для вариантов приложения, использующих технологии опроса и WebSocket

Поддержка спецификации HTML5 WebSocket браузерами

Как показано в табл. 6.1, на момент написания книги спецификация HTML5 WebSocket поддерживалась (или такая поддержка планировалась) рядом браузеров.

Поскольку данная спецификация поддерживается не всеми браузерами, то перед использованием веб-сокетов HTML5 целесообразно выполнить соответствующую проверку. О том, как это сделать программным путем, будет рассказано далее.

Таблица 6.1. Поддержка спецификации HTML5 WebSocket браузерами

Браузер	Описание
Chrome	Поддерживается в версиях 4+
Firefox	Поддерживается в версиях 4+
Internet Explorer	Поддержка отсутствует
Opera	Поддержка отсутствует
Safari	Поддерживается в версиях 5+

Простой эхо-сервер WebSocket

Прежде чем можно будет использовать протокол WebSocket, необходимо создать сервер, поддерживающий веб-сокеты. В этом разделе рассматривается, как создать простой эхо-сервер WebSocket. Для выполнения примеров, приведенных в этой главе, понадобится включить в программу простой сервер WebSocket, код которого написан на языке Python. Этот код находится в файле `code/websockets.py` на сайте книги.

Существующие серверы WebSocket

В настоящее время уже существует множество реализаций серверов WebSocket, а ряд других находится на стадии разработки. Ниже перечислены лишь некоторые из существующих серверов.

- **Kaazing WebSocket Gateway**. Шлюз WebSocket на основе Java.
- **`mod_pywebsocket`**. Расширение для HTTP-сервера Apache на основе Python.
- **Netty**. Сетевой фреймворк на основе Java, который включает поддержку протокола WebSocket.
- **`node.js`**. Серверный JavaScript-фреймворк, на основе которого было создано несколько серверов WebSocket.

Версия сервера Kaazing WebSocket Gateway включает полную поддержку эмуляции клиента WebSocket для браузеров, не имеющих собственной реализации WebSocket, что позволяет писать программы на основе протокола WebSocket уже сегодня, зная, что код будет работать во всех браузерах.

Чтобы запустить эхо-сервер Python WebSocket, к которому можно подключаться по адресу `ws://localhost:8080/echo`, откройте окно командной строки, перейдите в папку с файлом `websocket.py` и выполните следующую команду:

```
python websocket.py
```

В файлы примеров включен также файл с кодом *широковещательного* сервера (`code/broadcast.py`), подключение к которому осуществляется по адресу `ws://localhost:8080/broadcast`. В отличие от эхо-сервера, любое WebSocket-сообщение, отправленное этой реализации сервера, будет автоматически направляться *всем* адресатам, которые в настоящее время подключены к данному серверу. Это простой способ реализации широковещательной рассылки сообщений всем або-

нентам. Чтобы запустить широковещательный сервер, откройте окно командной строки, перейдите в папку с указанным файлом и выполните следующую команду:

```
python broadcast.py
```

Оба сценария используют библиотеку для работы с протоколом WebSocket, разработанную для этих примеров и включенную в файл `websocket.py`. Можете добавить пути к другим обработчикам событий, реализующим дополнительную функциональность на стороне сервера.

Примечание

Этот сервер предназначен только для работы по протоколу WebSocket и не может отвечать на HTTP-запросы. Поскольку установление соединения по протоколу WebSocket начинается с использования подмножества официального протокола HTTP и зависит от использования заголовка `Upgrade`, другие серверы могут обслуживать как запросы по протоколу WebSocket, так и запросы по протоколу HTTP через один и тот же порт.

Посмотрим, что происходит, когда браузер пытается обменяться данными с этим сервером. Когда браузер направляет запрос по URL-адресу веб-сокета, сервер отправляет обратно заголовки, завершающие обмен подтверждениями по протоколу WebSocket. Ответное квитирование WebSocket должно начинаться *строго* со строки `HTTP/1.1 101 WebSocket Protocol Handshake`. Фактически, порядок и содержание заголовков "рукопожатия" определяются более строго, чем для заголовков HTTP.

```
# записать заголовки ответов
self.send_bytes("HTTP/1.1 101 Web Socket Protocol Handshake\r\n")
self.send_bytes("Upgrade: WebSocket\r\n")
self.send_bytes("Connection: Upgrade\r\n")
self.send_bytes("Sec-WebSocket-Origin: %s\r\n" % headers["Origin"])
self.send_bytes("Sec-WebSocket-Location: %s\r\n" % head-
ers["Location"])

if "Sec-WebSocket-Protocol" in headers:
    protocol = headers["Sec-WebSocket-Protocol"]
    self.send_bytes("Sec-WebSocket-Protocol: %s\r\n" % protocol)

self.send_bytes("\r\n")
# записать хешированный маркер ответа
self.send_bytes(response_token)
```

Примечание

Если вы реализуете сервер WebSocket, обратитесь к черновику протокола на сайте IETF:

`http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol`

Либо найдите самую последнюю спецификацию протокола.

После завершения квитирования клиент и сервер могут в любой момент начать обмен сообщениями. Каждое соединение представляется на сервере экземпляром объекта `WebSocketConnection`. Приведенный ниже метод `send` этого объекта преобразует строку сообщения к виду, соответствующему протоколу WebSocket. Границы кадра обозначаются байтами `0x00` и `0xFF`, между которыми помещается строка сообщения в кодировке UTF-8. В нашем сервере каждое сообщение WebSocket является объектом `asyncore.dispatcher_with_send`, представляющим собой оболочку асинхронного сокета с поддержкой буферизации отправляемых данных.

Примечание

Существует множество других фреймворков ввода-вывода для языка Python и других языков программирования. Фреймворк Asyncore был выбран, поскольку он входит в стандартную библиотеку Python. Также заметьте, что данная реализации поддерживает как версию 75, так и версию 76 протокола WebSocket. Строго говоря, одновременная поддержка обоих версий не разрешается, однако наш пример предназначен для тестирования и используется исключительно в иллюстративных целях.

Объект `WebSocketConnection` наследуется от объекта `asyncore.dispatcher_with_send` и перекрывает метод `send`, добавляя возможность работать со строками в кодировке UTF-8 и кадрами строк WebSocket.

```
def send(self, s):
  if self.readystate == "open":
    self.send_bytes("\x00")
    self.send_bytes(s.encode("UTF8"))
    self.send_bytes("\xFF")
```

Обработчики для объектов `WebSocketConnection`, включенные в файл `websocket.py`, следуют упрощенному интерфейсу диспетчеризации. Метод обработчика `dispatch()` вызывается с передачей ему полезной нагрузки каждого из кадров, получаемых через соединение. Объект `EchoHandler` отправляет каждое сообщение обратно отправителю.

```
class EchoHandler(object):
    """
    Объект EchoHandler возвращает каждую поступающую строку
    тому же объекту WebSocket.
    """

    def __init__(self, conn):
        self.conn = conn

    def dispatch(self, data):
        self.conn.send("echo: " + data)
```

Базовый широковещательный сервер `broadcast.py` во многом работает точно так же, но в данном случае, когда обработчик получает строку, он отправляет сообщение на все подключенные веб-сокеты, как показано в следующем примере.

```
class BroadcastHandler(object):
    """
    Объект BroadcastHandler возвращает каждую поступающую строку
    каждому подключенному объекту WebSocket.
    """

    def __init__(self, conn):
        self.conn = conn

    def dispatch(self, data):
        for session in self.conn.server.sessions:
            session.send(data)
```

Обработчик в файле `broadcast.py` выполняет упрощенную широковещательную рассылку сообщений, которая обеспечивает лишь получение и отправку строк. Для нашего примера этого вполне достаточно. Однако вы должны понимать, что этой широковещательной службе недостает проверки поступающих данных, которая должна предусматриваться в реальных серверах сообщений. В коммерческом сервере WebSocket должна осуществляться по крайней мере проверка правильности форматирования входных данных.

В интересах цельности изложения в листингах 6.5 и 6.6 приведен полный код, содержащийся в файлах `websocket.py` и `broadcast.py`. Имейте в виду, что это всего лишь пример реализации сервера, непригодный для развертывания в производственных целях.

Листинг 6.5. Завершенный программный код для эхо-сервера (файл `websocket.py`)

```
#!/usr/bin/env python

import asyncore
import socket
import struct
import time
import hashlib

class WebSocketConnection(asyncore.dispatcher_with_send):
    def __init__(self, conn, server):
        asyncore.dispatcher_with_send.__init__(self, conn)

        self.server = server
        self.server.sessions.append(self)
        self.readystate = "connecting"
        self.buffer = ""

    def handle_read(self):
        data = self.recv(1024)
        self.buffer += data
        if self.readystate == "connecting":
            self.parse_connecting()
        elif self.readystate == "open":
            self.parse_frametype()

    def handle_close(self):
        self.server.sessions.remove(self)
        self.close()

    def parse_connecting(self):
        header_end = self.buffer.find("\r\n\r\n")
        if header_end == -1:
            return
        else:
            header = self.buffer[:header_end]
            # удалить заголовок и 4 байта конца строки из буфера
            self.buffer = self.buffer[header_end+4:]
            header_lines = header.split("\r\n")
            headers = {}

            # проверить HTTP-запрос и получить строку адреса
            method, path, protocol = header_lines[0].split(" ")
            if method != "GET" or protocol != "HTTP/1.1" or
               path[0] != "/":
                self.terminate()
                return

            # проанализировать заголовки
            for line in header_lines[1:]:
                key, value = line.split(": ")
                headers[key] = value

            headers["Location"] = "ws://" + headers["Host"] + path
```

```
        self.readystate = "open"
        self.handler = self.server.handlers.get(path, None)(self)

        if "Sec-WebSocket-Key1" in headers.keys():
            self.send_server_handshake_76(headers)
        else:
            self.send_server_handshake_75(headers)

    def terminate(self):
        self.ready_state = "closed"
        self.close()

    def send_server_handshake_76(self, headers):
        """
        Послать ответное подтверждение WebSocket Protocol v.76
        """

        key1 = headers["Sec-WebSocket-Key1"]
        key2 = headers["Sec-WebSocket-Key2"]
        # прочитать дополнительные 8 байт из буфера
        key3, self.buffer = self.buffer[:8], self.buffer[8:]

        response_token = self.calculate_key(key1, key2, key3)

        # записать заголовки ответов
        self.send_bytes("HTTP/1.1 101 Web Socket Protocol
                        Handshake\r\n")
        self.send_bytes("Upgrade: WebSocket\r\n")
        self.send_bytes("Connection: Upgrade\r\n")
        self.send_bytes("Sec-WebSocket-Origin: %s\r\n" %
                        headers["Origin"])
        self.send_bytes("Sec-WebSocket-Location: %s\r\n" %
                        headers["Location"])

        if "Sec-WebSocket-Protocol" in headers:
            protocol = headers["Sec-WebSocket-Protocol"]
            self.send_bytes("Sec-WebSocket-Protocol: %s\r\n" %
                            protocol)

        self.send_bytes("\r\n")
        # записать хешированный маркер ответа
        self.send_bytes(response_token)

    def calculate_key(self, key1, key2, key3):
        # проанализировать ключи 1 и 2 путем извлечения
        # числовых символов
        num1 = int("".join([digit for digit in list(key1) if
                    digit.isdigit()]))
        spaces1 = len([char for char in list(key1) if char == " "])
        num2 = int("".join([digit for digit in list(key2) if
                    digit.isdigit()]))
        spaces2 = len([char for char in list(key2) if char == " "])

        combined = struct.pack(">II", num1/spaces1, num2/spaces2) +
                    key3
        # md5 возвращает сумму байтов в объекте combined
        return hashlib.md5(combined).digest()

    def send_server_handshake_75(self, headers):
        """
        Послать ответное подтверждение WebSocket Protocol v.75
        """
```

```
        self.send_bytes("HTTP/1.1 101 Web Socket Protocol
                        Handshake\r\n")
        self.send_bytes("Upgrade: WebSocket\r\n")
        self.send_bytes("Connection: Upgrade\r\n")
        self.send_bytes("WebSocket-Origin: %s\r\n" %
                        headers["Origin"])
        self.send_bytes("WebSocket-Location: %s\r\n" %
                        headers["Location"])
        if "Protocol" in headers:
            self.send_bytes("WebSocket-Protocol: %s\r\n" %
                            headers["Protocol"])
        self.send_bytes("\r\n")
    def parse_frametype(self):
        while len(self.buffer):
            type_byte = self.buffer[0]
            if type_byte == "\x00":
                if not self.parse_textframe():
                    return
    def parse_textframe(self):
        terminator_index = self.buffer.find("\xFF")
        if terminator_index != -1:
            frame = self.buffer[1:terminator_index]
            self.buffer = self.buffer[terminator_index+1:]
            s = frame.decode("UTF8")
            self.handler.dispatch(s)
            return True
        else:
            # неполный фрейм
            return false
    def send(self, s):
        if self.readystate == "open":
            self.send_bytes("\x00")
            self.send_bytes(s.encode("UTF8"))
            self.send_bytes("\xFF")
    def send_bytes(self, bytes):
        asyncore.dispatcher_with_send.send(self, bytes)
class EchoHandler(object):
    """
    Объект EchoHandler возвращает каждую поступающую строку
    тому же объекту WebSocket.
    """
    def __init__(self, conn):
        self.conn = conn
    def dispatch(self, data):
        self.conn.send("echo: " + data)
class WebSocketServer(asyncore.dispatcher):
    def __init__(self, port=80, handlers=None):
        asyncore.dispatcher.__init__(self)
        self.handlers = handlers
        self.sessions = []
        self.port = port
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.set_reuse_addr()
        self.bind(("", port))
        self.listen(5)

    def handle_accept(self):
        conn, addr = self.accept()
        session = WebSocketConnection(conn, self)
```

```
if __name__ == "__main__":
    print "Starting WebSocket Server"
    WebSocketServer(port=8080, handlers={"/echo": EchoHandler})
    asyncore.loop()
```

Возможно, вы обратили внимание на необычную процедуру вычисления ключа при выполнении "рукопожатия" по протоколу WebSocket. Это делается с целью предотвращения межпротокольных атак. Если говорить коротко, то это должно пресекать попытки подделки соединений с серверами, работающими не по протоколу WebSocket, со стороны злонамеренного клиентского кода WebSocket. На момент существования черновика-76 спецификации эта часть процедуры квитирования все еще подлежала обсуждению.

Листинг 6.6. Завершенный программный код для широковещательного сервера (файл `broadcast.py`)

```
#!/usr/bin/env python

import asyncore
from websocket import WebSocketServer

class BroadcastHandler(object):
  """
  Объект BroadcastHandler возвращает каждую поступающую строку
  каждому подключенному объекту WebSocket.
  """

  def __init__(self, conn):
  self.conn = conn

  def dispatch(self, data):
  for session in self.conn.server.sessions:
    session.send(data)

  if __name__ == "__main__":
    print "Starting WebSocket broadcast server"
    WebSocketServer(port=9999, handlers={"/broadcast": BroadcastHan-
dler})
      asyncore.loop()
```

Теперь, когда у нас имеется работающий эхо-сервер, необходимо создать код клиентской части.

Программный интерфейс HTML5 WebSocket

В этом разделе программный интерфейс HTML5 WebSocket рассматривается более подробно.

Проверка поддержки в браузере

Прежде чем использовать функции HTML5 WebSocket, следует убедиться, что браузер предоставляет необходимую поддержку. Если такая поддержка отсутствует, вы сможете предоставить замещающий текст, предлагающий пользователям вашего приложения установить более новую версию браузера. Один из возможных способов тестирования поддержки в браузере демонстрируется в листинге 6.7.

Листинг 6.7. Проверка поддержки в браузере

```
function loadDemo() {
  if (window.WebSocket) {
    document.getElementById("support").innerHTML =
            "HTML5 WebSocket поддерживается вашим браузером.";
  } else {
     document.getElementById("support").innerHTML =
            "HTML5 WebSocket не поддерживается вашим браузером.";
  }
}
```

В этом примере проверка осуществляется функцией `loadDemo()`, которую можно вызывать при загрузке страницы. Вызов `window.WebSocket` вернет объект `WebSocket`, если он существует, или передаст управление другой ветви кода. В обоих случаях для индикации того, предоставляет или не предоставляет браузер необходимую поддержку, элемент `support`, определенный ранее на странице, обновляется соответствующим сообщением.

Для проверки предоставляемой браузером поддержки можно воспользоваться консолью браузера (например, Firebug или Chrome Developer Tools). Рис. 6.5 иллюстрирует, как проверить наличие встроенной поддержки веб-сокетов в браузере Google Chrome (если таковая отсутствует, вызов `window.WebSocket` вернет строку "undefined").

Использование базовых функций

Код приведенных ниже примеров доступен на веб-сайте книги в папке `code/websockets`. В этой папке содержатся файлы `websocket.html` и `broadcast.html` (вместе с дополнительным файлом `tracker.html`, который используется в следующем разделе), а также представленный ранее код сервера WebSocket, который запускается в среде Python.

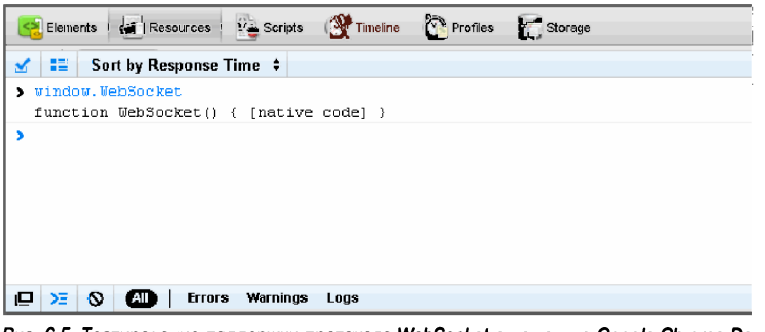


Рис. 6.5. Тестирование поддержки протокола WebSocket с помощью Google Chrome Developer Tools

Создание объекта `WebSocket` и подключение к серверу

Использование интерфейса `WebSocket` не представляет сложностей. Чтобы установить соединение, достаточно создать новый экземпляр объекта `WebSocket` и предоставить новому объекту URL-адрес конечного узла, с которым необходимо установить соединение. Для подключения к веб-сокетам по обычному и безопасному соединению следует использовать соответственно префиксы `ws://` и `wss://`.

```
url = "ws://localhost:8080/echo";
w = new WebSocket(url);
```

Добавление обработчиков событий

Программирование веб-сокетов следует модели программирования асинхронных соединений; как только открыт сокет, остается лишь ожидать наступления соответствующих событий. Больше нет необходимости самостоятельно опрашивать сервер. Для этого вы должны добавить в объект `WebSocket` функции обратного вызова, перехватывающие события.

Объект `WebSocket` перехватывает три события: `open`, `close` и `message`. Событие `open` запускается при установлении соединения, событие `message` — при получении сообщений, а событие `close` — при закрытии соединения с веб-сокетом. Как и в большинстве библиотек JavaScript, имеются соответствующие функции обратного вызова (`onopen`, `onmessage` и `onclose`), которые запускаются при получении событий.

```
w.onopen = function() {
  log("открыто");
  w.send("Работа по протоколу WebSocket...");
}
w.onmessage = function(e) {
  log(e.data);
}
w.onclose = function(e) {
  log("закрыто");
}
```

Отправка сообщений

Все время, пока сокет открыт (т.е. после вызова обработчика событий `onopen` и до вызова обработчика событий `onclose`), можно использовать метод `send` для отправки сообщений. Кроме того, после отправки сообщения можно разорвать соединение, вызвав метод `close`, либо оставить его открытым.

```
document.getElementById("sendButton").onclick = function() {
  w.send(document.getElementById("inputMessage").value);}
```

Вот и все. Нам удалось очень легко организовать двухсторонний канал связи с браузером. В интересах цельности изложения в листинге 6.8 приведен полный текст HTML-страницы вместе с клиентским кодом веб-сокета.

Листинг 6.8. Код страницы `websocket.html`

```
<!DOCTYPE html>
<title>Тестирование WebSocket</title>
<script>

    var log = function(s) {
        if (document.readyState !== "complete") {
            log.buffer.push(s);
        } else {
            document.getElementById("output").innerHTML += (s + \n");
        }
    }
    log.buffer = [];

    url = "ws://localhost:8080/echo";
    w = new WebSocket(url);
    w.onopen = function() {
        log("открыто");
        w.send("Работа по протоколу WebSocket...");
    }
```

```
    w.onmessage = function(e) {
        log(e.data);
    }
    w.onclose = function(e) {
        log("закрыто");
    }

    window.onload = function() {
        log(log.buffer.join("\n"));
        document.getElementById("sendButton").onclick = function() {
            w.send(document.getElementById("inputMessage").value);
        }
    }
</script>

<input type="text" id="inputMessage" value="Привет, Web Socket!">
<button id="sendButton">Отправить</button>
<pre id="output"></pre>
```

Тестирование страницы WebSocket

Чтобы протестировать страницу `websocket.html`, содержащую код WebSocket, откройте окно командной строки, перейдите в соответствующую папку и выполните следующую команду:

```
python -m SimpleHTTPServer 9999
```

После этого откройте еще одно окно командной строки, перейдите к папке, содержащей код WebSocket, и запустите сервер Python WebSocket, выполнив следующую команду:

```
python websocket.py
```

Наконец, откройте браузер, поддерживающий веб-сокеты, и откройте в нем страницу `http://localhost:9999/websocket.html`.

Результат открытия веб-страницы представлен на рис. 6.6.

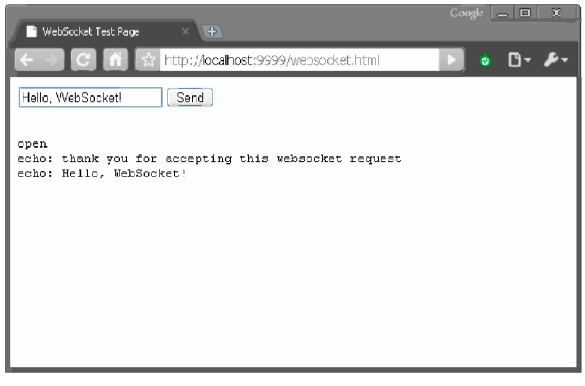


Рис. 6.6. Страница `websocket.html`

В папке с кодом примеров содержится также веб-страница, которая подключается к широковещательной службе, созданной в предыдущем разделе. Чтобы увидеть это в действии, закройте окно командной строки, в котором выполняется сервер WebSocket, перейдите в папку, содержащую код WebSocket, и выполните следующую команду для запуска сервера Python WebSocket:

```
python broadcast.py
```

Откройте два отдельных браузера, имеющих встроенную поддержку WebSocket, и перейдите (в каждом из браузеров) на страницу по такому адресу:

```
http://localhost:9999/broadcast.html
```

Результаты работы широковещательного сервера WebSocket в виде отдельных веб-страниц представлены на рис. 6.7.

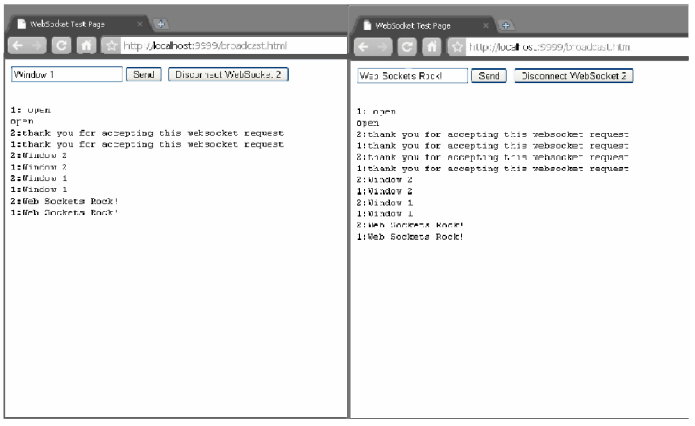


Рис. 6.7. Результаты работы страницы `broadcast.html`*, наблюдаемые в двух браузерах*

Создание приложения на основе веб-сокетов HTML5

Теперь, когда вы уже ознакомились с основами функционирования веб-сокетов, настало время создать нечто более существенное. Ранее мы использовали программный интерфейс HTML5 Geolocation для создания приложения, которое позволяло вычислять пройденное расстояние непосредственно с помощью веб-страницы. Мы можем обратиться к предыдущему опыту создания геолокационного приложения, объединив его с новой поддержкой веб-сокетов для создания простого приложения, к которому могут подключаться несколько участников: определителя местоположения.

Примечание

Мы будем использовать описанный ранее широковещательный сервер, и поэтому, если вы не ознакомились с предыдущим материалом, потратьте некоторое время на его изучение.

Приложение, которое мы собираемся создать, объединит в себе поддержку спецификаций WebSocket и Geolocation для определения местоположения пользователя и рассылки полученных данных всем подключившимся абонентам. Компьютеры всех пользователей, которые загрузят это приложение и подключатся к одному и тому же широковещательному серверу, будут регулярно отправлять ему данные о своем географическом местоположении, используя веб-сокеты. В то же время приложение будет перехватывать любые сообщения, поступающие от сервера, и обновлять в режиме реального времени отображаемые данные для всех подключенных абонентов. В сценарии с марафоном такое приложение может информировать участников забега о местоположении их соперников, позволяя им решать, бежать им быстрее или медленнее.

Это небольшое приложение не включает никакой иной персональной информации, кроме географической широты и долготы точки, в которой находится участник соревнований. Его имя, дата рождения и название любимого мороженого остаются в строгом секрете.

Вас предупредили!

Говорит Брайан: "В этом приложении многое связано с предоставлением вашей личной информации в общее пользование. Конечно, в данном случае речь идет только о местоположении. Однако если вы (или ваши пользователи) не до конца осознали, в чем состоял смысл предупреждения, отображавшегося в браузере при первой попытке доступа к функциям Geolocation API, то данное приложение должно отчетливо продемонстрировать вам, как легко критически важные данные могут быть переданы удаленным узлам. Поэтому следует обязательно убедиться в том, что пользователи отдают себе полный отчет относительно последствий отправки ими своих геолокационных данных.

В случае любых сомнений информируйте пользователей о том, каким образом могут быть использованы их личные данные. Сделайте отказ от предоставления своих данных в общее пользование самой доступной опцией.

Однако довольно предупреждений. Займемся конкретным кодом. Как обычно, полный код примера предоставляется на сайте книги. Здесь же мы сконцентрируем свое внимание на наиболее важных его частях. Результаты работы готового приложения представлены на рис. 6.8.

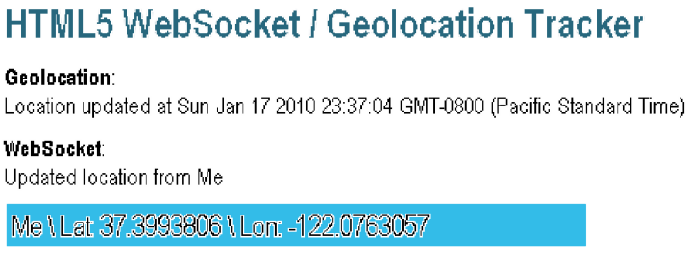


Рис. 6.8. Результаты работы приложения, определяющего местоположение пользователя

Создание HTML-кода

HTML-разметка для нашего приложения намеренно выбрана простой, чтобы можно было сфокусироваться на обработке данных. Насколько она проста, судите сами.

```
<body onload="loadDemo()">

<h1>HTML5 WebSocket / Geolocation Tracker</h1>

<div><strong>Geolocation</strong>: <p id="geoStatus">HTML5
Geolocation is <strong>not</strong> supported in your browser.</p>
</div>
<div><strong>WebSocket</strong>: <p id="socketStatus">WebSockets
are <strong>not</strong> supported in your browser.</p></div>

</body>
```

Мы включили в разметку только заголовок приложения и две строки состояния: одна из них информирует пользователя об обновлении геолокационных данных, а вторая — об активности веб-сокета. Фактические данные о местоположении будут помещаться на страницу в виде сообщений, получаемых в режиме реального времени.

Сообщения о состоянии приложения по умолчанию соответствуют отсутствию поддержки спецификаций Geolocation и WebSocket в браузере пользователя. Обнаружение поддержки любой из этих технологий приводит к соответствующему обновлению информации о состоянии.

```
<script>

    // Ссылка на WebSocket
    var socket;

    // Псевдослучайный ID для этого сеанса
    var myId = Math.floor(100000*Math.random());

    // Количество отображаемых в настоящее время строк данных
    var rowCount = 0;
```

Функциональная часть приложения реализуется посредством сценария. Прежде всего мы создаем несколько переменных, описанных ниже.

- Глобальная переменная `socket`, с помощью которой функции будут получать доступ к веб-сокету.
- Переменная `myId`, являющаяся случайным числом, принимающим значения в диапазоне от `0` до `100000`, которые идентифицируют геолокационные данные в сети. Это число используется для привязки координат местоположения, испытывающих изменения во времени, к их источнику без использования более персонализированных данных, таких, например, как имена. Ввиду того, что используемый набор псевдослучайных чисел достаточно велик, вероятность того, что разным пользователям будет присвоен один и тот же идентификатор, крайне мала.
- Переменная `rowCount`, в которой хранится число уникальных пользователей, передающих нам свои геолокационные данные. Эта переменная используется главным образом для обеспечения подходящего визуального форматирования информации.

Следующие две функции должны быть вам уже знакомы. Как и в других примерах приложений, мы предоставляем средства, облегчающие обновление сообщений о состоянии приложения. На этот раз нам необходимо позаботиться об обновлении двух таких сообщений.

```
function updateSocketStatus(message) {
    document.getElementById("socketStatus").innerHTML = message;
}

function updateGeolocationStatus(message) {
    document.getElementById("geoStatus").innerHTML = message;
}
```

Всегда неплохо предусмотреть дружественный по отношению к пользователю набор сообщений об ошибках, которые выводятся, если при получении координат местоположения возникают какие-либо проблемы. Если вам требуется более подробная информация относительно обработки ошибок, связанных с геолокационной частью приложения, обратитесь к главе 4.

```
function handleLocationError(error) {
    switch(error.code)
    {
        case 0:
          updateGeolocationStatus("There was an error while
                                   retrieving your location: " +
                                   error.message);
          break;
        case 1:
          updateGeolocationStatus("The user prevented this page
                                   from retrieving a location.");
          break;
        case 2:
          updateGeolocationStatus("The browser was unable to
                                   determine your location: " +
                                   error.message);
          break;
        case 3:
          updateGeolocationStatus("The browser timed out before
                                   retrieving the location.");
          break;
    }
}
```

Добавление кода для работы с веб-сокетом

Перейдем к рассмотрению более важных вопросов. Функция `loadDemo` вызывается во время начальной загрузки страницы, что делает ее отправной точкой нашего приложения.

```
function loadDemo() {
    // Убедиться в наличии поддержки WebSocket
    if (window.WebSocket) {

       // Указать расположение нашего широковещательного
       // сервера WebSocket
       url = "ws://localhost:8080";
       socket = new WebSocket(url);
       socket.onopen = function() {
          updateSocketStatus("Connected to WebSocket tracker
                              server");
       }
       socket.onmessage = function(e) {
          updateSocketStatus("Updated location from " +
                              dataReturned(e.data));
       }
}
```

Первое, что мы здесь делаем, — это устанавливаем WebSocket-соединение. Как и во всем, что касается использования технологии HTML5, прежде чем что-либо предпринимать, целесообразно сначала убедиться в наличии необходимой поддержки, в связи с чем мы проверяем, поддерживается ли объект `window.WebSocket` данным браузером.

Как только проверка выполнена, мы устанавливаем соединение с удаленным широковещательным сервером, используя описанный ранее формат строки подключения. Для сохранения информации о соединении используется глобальная переменная `socket`.

Наконец, мы объявляем два обработчика для выполнения определенных действий при изменении состояния веб-сокета. Обработчик событий onopen просто обновляет сообщение о состоянии приложения, информируя пользователя об успешном установлении соединения. Точно так же обработчик событий onmessage информирует пользователя о получении сообщения. Кроме того, этот обработчик вызывает функцию dataReturned, отображающую поступившие данные на странице, о чем пойдет речь далее.

Добавление кода для работы с геолокационными данными

Приведенный ниже код должен быть вам знаком из главы 4. Здесь осуществляется проверка поддержки службы Geolocation и соответствующим образом изменяется сообщение о состоянии приложения.

```
        var geolocation;
        if(navigator.geolocation) {
            geolocation = navigator.geolocation;
            updateGeolocationStatus("HTML5 Geolocation is supported
                                     in your browser.");
        }

        // Зарегистрировать обработчик для обновления позиции
        // с помощью Geolocation API
        geolocation.watchPosition(updateLocation,
                                  handleLocationError,
                                  {maximumAge:20000});
    }
```

Как и до этого, мы отслеживаем изменения текущего местоположения и регистрируем функцию updateLocation, которая будет вызываться всякий раз, когда происходят такие изменения. Ошибки обрабатываются функцией handleLocationError, а данные о местоположении считаются устаревшими по прошествии двадцати секунд.

Следующий раздел кода представляет собой обработчик, который вызывается браузером всякий раз, когда становятся доступными данные о новом местоположении.

```
function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var timestamp = position.timestamp;
    updateGeolocationStatus("Location updated at " + timestamp);
    // Отправить данные о местоположении через WebSocket
    var toSend = JSON.stringify([myId, latitude, longitude]);
    sendMyLocation(toSend);
}
```

Этот код аналогичен коду соответствующего обработчика из главы 4, хотя и проще его. Сначала мы извлекаем географические широту и долготу, а также временную метку из данных о позиции, предоставленных браузерами, а затем изменяем сообщение о состоянии приложения для отражения того факта, что получены новые данные.

Сводим все вместе

Последний фрагмент кода отвечает за подготовку строки сообщения к отправке на удаленный сервер WebSocket. Данные в этой строке кодируются в формате JSON.

```
"[<id>, <широта>, <долгота>]"
```

Идентификатор ID — это случайное значение, уже созданное ранее для идентификации данного пользователя. Значения широты и долготы предоставляются объектом, хранящим данные о географическом местоположении пользователя. Сообщение отправляется непосредственно серверу в виде строки в формате JSON.

Собственно код, осуществляющий отправку данных о местоположении серверу, находится в функции `sendMyLocation()`.

```
function sendMyLocation(newLocation) {
    if (socket) {
       socket.send(newLocation);
    }
}
```

Если сокет успешно создан (и сохранен для последующего доступа к нему), можно отправить строку сообщения, переданную данной функции, на сервер. Как только эти данные поступят, сервер широковещательной рассылки сообщений отправит строку с данными о местоположении всем браузерам, которые в настоящее время подключены к серверу и ожидают прихода сообщений. Таким образом, каждый участник соревнований будет знать о том, где вы находитесь.

Организовав отправку сообщений, посмотрим, как эти сообщения должны обрабатываться при их поступлении в браузер. Вспомните, что ранее мы зарегистрировали обработчик событий `onmessage` в сокете для передачи поступающих сообщений функции `dataReturned()`. Рассмотрим теперь более подробно окончательный вид этой функции.

```
function dataReturned(locationData) {
    // Разбить данные на ID, широту и долготу
    var allData = JSON.parse(locationData);
    var incomingId   = allData[1];
    var incomingLat  = allData[2];
    var incomingLong = allData[3];
```

Функция `dataReturned()` решает две задачи. Она создает (или обновляет) элемент, отображающий на странице данные о позиции, содержащиеся в пришедшей строке сообщения, а затем возвращает текст, представляющий пользователя, от которого поступило это сообщение. Это имя пользователя будет выводиться в верхней части страницы вызывающей функцией, т.е. обработчиком `socket.onmessage`.

Первое, что делает функция обработки данных, — разбивает поступившее сообщение на исходные компоненты с помощью метода `JSON.parse`. Хотя для повышения надежности приложения следовало бы осуществлять проверку правильности форматирования данных, мы будем предполагать, что все сообщения, приходящие на сервер, отформатированы надлежащим образом, и поэтому из строки просто выделяются случайный идентификатор ID, широта и долгота.

```
    // Найти HTML-элемент, соответствующий данному ID.
    // Если такой элемент не существует, создать его.
    var incomingRow = document.getElementById(incomingId);
    if (!incomingRow) {
        incomingRow = document.createElement('div');
        incomingRow.setAttribute('id', incomingId);
```

Наш демонстрационный пользовательский интерфейс будет создавать видимый элемент `<div>` для всех случайных идентификаторов ID, которым соответствуют получаемые сообщения. В том числе это касается и идентификатора самого пользователя. Иными словами, в браузере пользователя будут отображаться и его собственные данные, после того как они были отправлены широковещательному серверу WebSocket и возвращены им.

Согласно вышесказанному, первое, что необходимо сделать с идентификатором, извлеченным из строки сообщения, — это найти соответствующий ему отображаемый элемент. Если таковой пока еще не существует, мы создаем его и устанавливаем для его атрибута `id` значение, равное значению `id`, возвращенному сервером сокета, для будущего использования.

```
            incomingRow.userText = (incomingId == myId) ?
                                        'Me'            :
                                        'User ' + rowCount;

            rowCount++
```

Выяснить, какой текст следует отобразить в строке данных, не составляет труда. Если полученное значение ID совпадает со значением ID пользователя, отображается строка `'me'`. В противном случае имя пользователя будет представлять собой комбинацию общей для всех строки и инкрементируемого значения счетчика строк.

```
            document.body.appendChild(incomingRow);
        }
```

Подготовленный элемент отображения вставляется в конец страницы. Независимо от того, является ли данный элемент вновь созданным или он уже существовал (в силу того, что обновление местоположения для данного конкретного пользователя было произведено не впервые), отображаемая строка данных должна быть обновлена текущей текстовой информацией.

```
    // Обновить текст в строке новыми значениями
    incomingRow.innerHTML = incomingRow.userText + " \\ Lat: " +
                            incomingLat + " \\ Lon: " +
                            incomingLong;

    return incomingRow.userText;
}
```

В нашем случае мы отделяем текстовое имя пользователя от численных значений широты и долготы символами обратной косой черты (разумеется, маскированными надлежащим образом). И наконец, отображаемое имя возвращается вызывающей функции для обновления строки состояния.

На этом создание составного приложения на базе технологий WebSocket и Geolocation завершено. Испытайте его в работе, имея при этом в виду, что понаблюдать за достаточно большим количеством обновлений вам удастся лишь в том случае, если к приложению обратятся одновременно несколько браузеров. В качестве упражнения попробуйте усовершенствовать этот пример, предусмотрев в нем отображение поступающих данных о местоположении на карте Google Maps.

Финальный код приложения

В интересах цельности изложения в листинге 6.9 представлен полный код приложения, содержащийся в файле `tracker.html`.

Листинг 6.9. Код, содержащийся в файле `tracker.html`

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

<head>
<title>HTML5 WebSocket / Geolocation Tracker</title>
<link rel="stylesheet" href="styles.css">
</head>
```

```
<body onload="loadDemo()">
<h1>HTML5 WebSocket / Geolocation Tracker</h1>

<div><strong>Geolocation</strong>: <p id="geoStatus">HTML5
Geolocation is <strong>not</strong> supported in your browser.</p>
</div>
<div><strong>WebSocket</strong>: <p id="socketStatus">WebSockets
are <strong>not</strong> supported in your browser.</p></div>

<script>

    // Ссылка на WebSocket
    var socket;

    // Псевдослучайный ID для этого сеанса
    var myId = Math.floor(100000*Math.random());

    // Количество отображаемых в настоящее время строк данных
    var rowCount = 0;

    function updateSocketStatus(message) {
        document.getElementById("socketStatus").innerHTML = message;
    }

    function updateGeolocationStatus(message) {
        document.getElementById("geoStatus").innerHTML = message;
    }

    function handleLocationError(error) {
        switch(error.code)
        {
        case 0:
          updateGeolocationStatus("There was an error while
                                   retrieving your location: " +
                                   error.message);
          break;
        case 1:
          updateGeolocationStatus("The user prevented this page
                                   from retrieving a
                                   location.");
          break;
        case 2:
          updateGeolocationStatus("The browser was unable to
                                   determine your location: " +
                                   error.message);
          break;
        case 3:
          updateGeolocationStatus("The browser timed out before
                                   retrieving the location.");
          break;
        }
    }

    function loadDemo() {
        // убедиться в наличии поддержки объекта WebSocket
        if (window.WebSocket) {
            // Указать расположение сервера WebSocket
```

```
        url = "ws://localhost:8080";
        socket = new WebSocket(url);
        socket.onopen = function() {
            updateSocketStatus("Connected to WebSocket tracker
                                server");
        }
        socket.onmessage = function(e) {
            updateSocketStatus("Updated location from " +
                                dataReturned(e.data));
        }
    }

    var geolocation;
    if(navigator.geolocation) {
        geolocation = navigator.geolocation;
        updateGeolocationStatus("HTML5 Geolocation is supported
                                 in your browser.");
    }

    // зарегистрировать обработчик для обновления позиции
    // с помощью Geolocation API
    geolocation.watchPosition(updateLocation,
                              handleLocationError,
                              {maximumAge:20000});
}

function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var timestamp = position.timestamp;

    updateGeolocationStatus("Location updated at " + timestamp);

    // отправить данные о местоположении через веб-сокет
    var toSend = JSON.stringify([myId, latitude, longitude]);
    sendMyLocation(toSend);
}

function sendMyLocation(newLocation) {
    if (socket) {
        socket.send(newLocation);
    }
}

function dataReturned(locationData) {
    // разбить данные на ID, широту и долготу
    var allData = JSON.parse(locationData)
    var incomingId   = allData[1];
    var incomingLat  = allData[2];
    var incomingLong = allData[3];

    // Найти HTML-элемент, соответствующий данному ID.
    // Если такой элемент не существует, создать его.
    var incomingRow = document.getElementById(incomingId);
    if (!incomingRow) {
        incomingRow = document.createElement('div');
        incomingRow.setAttribute('id', incomingId);
```

```
            incomingRow.userText = (incomingId == myId) ?
                                        'Me'            :
                                        'User ' + rowCount;

            rowCount++;

            document.body.appendChild(incomingRow);
        }

        // обновить строку текста новыми значениями
        incomingRow.innerHTML = incomingRow.userText + " \\ Lat: " +
                                incomingLat + " \\ Lon: " +
                                incomingLong;

        return incomingRow.userText;
    }

</script>
</body>
</html>
```

Резюме

Как было показано в этой главе, протокол HTML5 WebSocket реализует простой, но вместе с тем мощный механизм создания впечатляющих приложений, работающих в реальном времени.

Сначала мы рассмотрели природу самого протокола и его взаимодействие с существующим HTTP-трафиком. Были сопоставлены требования к служебному сетевому трафику текущих коммуникационных приложений и протокола WebSocket.

Чтобы проиллюстрировать работу веб-сокетов, мы исследовали простую реализацию сервера WebSocket, с помощью которой показали, насколько просто этот протокол реализуется на практике. Аналогичным образом, мы исследовали клиентские функции WebSocket API, обращая ваше внимание на обеспечиваемую ими легкость интеграции с существующими методами программирования на JavaScript.

Наконец, мы проанализировали пример более сложного приложения, объединяющего в себе мощь технологий Geolocation и WebSocket, чтобы продемонстрировать, насколько плодотворным может быть одновременное использование обеих технологий.

Теперь, когда вы узнали, как в HTML5 используются элементы сетевого программирования на основе сокетов, мы можем заняться сбором более интересных данных, нежели просто текущее местоположение пользователя. В следующей главе мы рассмотрим усовершенствования, затронувшие элементы управления форм в HTML5.

Глава 7

Работа с формами в HTML5

Эта глава посвящена рассмотрению новых возможностей, которые предоставляет нам технология-долгожитель — формы HTML. С момента своего появления формы находились в эпицентре бурного информационного взрыва, положившего начало Интернету. Не будь форм, многое из того, что сейчас является непременными атрибутами современной жизни, — интернет-магазины, социальные сети, возможности эффективного поиска информации, — было бы просто невозможно.

К сожалению, несмотря на то что работа над спецификацией HTML5 Forms длится уже много лет, как сама спецификация, так и ее реализации далеки от завершения. Хорошая новость состоит в том, что при всей незначительности видимых успехов прогресс в этой области значительно ускорился. Однако есть и плохая новость. Спецификация HTML5 Forms поддерживается разными браузерами в разной степени, и для определения ее подмножества, которое будет работать в ваших целевых браузерах, придется немного потрудиться. Спецификация форм детализирует большой набор классов и функций, и нередко оказывается так, что в каждую новую версию HTML5-совместимого браузера добавляется поддержка одного или нескольких элементов управления и некоторых новых полезных средств верификации документов.

Как бы то ни было, эта глава поможет вам проложить правильный курс в виртуальном море элементов управления и определить, какие из них готовы к использованию уже сегодня, а какие пока только на горизонте.

Обзор форм HTML5

Если вы уже знакомы с формами HTML, — а мы полагаем, что это действительно так, раз вы заинтересованы в профессиональном овладении навыками HTML-программирования, — то обнаружите, что новые дополнения HTML5 аккуратно ложатся на существующий прочный фундамент. В отличие от XForms, вас должно радовать то, что:

- формы по-прежнему должны инкапсулироваться в элементы `<form>`, в которых устанавливаются основные атрибуты отправки;
- формы по-прежнему отправляют значения элементов управления серверу, когда пользователь или приложение отправляют страницу;
- все знакомые вам элементы управления форм — текстовые поля, переключатели, флажки и т.п. — остались доступными и работают точно так же, как и прежде (хотя и предлагают некоторые новые возможности);
- для тех, кто хочет создавать собственные модификаторы и обработчики, к элементам управления форм по-прежнему возможен доступ из сценариев.

Сравнение спецификаций HTML Forms и XForms

Возможно, на протяжении последних нескольких лет, задолго до того, как разработка спецификации HTML5 набрала заметный темп, вам приходилось время от времени слышать упоминания об XForms. Это основанный на XML, мощный, хотя и несколько сложный стандарт, описывающий поведение форм на стороне клиента, который разрабатывался отдельной рабочей группой Консорциума W3C в течение примерно десяти лет. Стандарт XForms в полной мере использует мощные возможности стандарта XML Schema для определения точных правил проверки достоверности (валидации) и правильности форматирования документов. К сожалению, ни один из имеющихся в настоящее время основных браузеров не поддерживает XForms без привлечения дополнительных подключаемых модулей.

Функциональные формы

В отличие от XForms, спецификация HTML5 Forms фокусируется на развитии существующих обычных форм HTML, стремясь расширить круг элементов управления, охватываемых формами, и преодолеть ограничения практического характера, с которыми сегодня приходится сталкиваться разработчикам. Однако следует сделать одно важное замечание, о котором нельзя забывать, особенно сравнивая реализации форм в различных браузерах.

Примечание

Чрезвычайно важно, чтобы вы понимали: спецификация HTML5 Forms имеет дело с функциональным поведением и семантикой форм, а не с их внешним видом и особенностями отображения.

К примеру, в то время как спецификация содержит детальные описания классов и функций для работы с элементами, предназначенными для выбора цвета и даты, числовыми селекторами, адресами электронной почты и т.п., в ней ни слова не говорится о том, как эти элементы должны визуализироваться для конечного пользователя. Благодаря этому браузеры получают большую свободу действий на многих уровнях. Это предоставляет им возможность соревноваться во внедрении новейших способов интерактивного взаимодействия с пользователем, а кроме того, отделяет стилевое оформление от семантики и обеспечивает возможность такого взаимодействия со специализированными или будущими устройствами пользовательского ввода, которое лучше всего соответствует их естественным особенностям функционирования. Однако до тех пор пока платформы целевых браузеров не будут поддерживать все элементы управления форм, которые используются в вашем приложении, обязательно предоставляйте достаточно полную контекстную информацию, чтобы пользователи не терялись в догадках относительно того, какие альтернативные возможности управления формой вы им предлагаете. При наличии соответствующих подсказок и описаний пользователи не будут испытывать трудностей в процессе работы с вашим приложением, даже если оно будет вынуждено отображать альтернативное содержимое в случае неизвестных для браузера типов элементов ввода.

Спецификация HTML5 Forms охватывает большое количество новых классов и типов элементов, поддержке которых сейчас уделяется самое пристальное внимание. Чтобы нам было легче разбираться со всеми особенностями новой функциональности, разобьем ее на две категории:

- новые типы элементов ввода;
- новые функции и атрибуты.

Но прежде чем двигаться дальше, целесообразно кратко проанализировать, какой уровень поддержки спецификации HTML5 Forms обеспечивают современные браузеры.

Поддержка спецификации HTML5 Forms браузерами

Поддержка спецификации HTML5 Forms браузерами расширяется, хотя и остается ограниченной. Создается впечатление, что многие производители браузеров, не считая Opera, пока что не прилагали заметных усилий для разработки собственных новых элементов управления. Однако недавно в браузерах WebKit поддержка форм была улучшена — возможно, в связи с появлением iPhone и его виртуальной дисплейной клавиатуры, о чем вскоре будет говориться. В настоящее время внедряются такие усовершенствованные инструменты, как, например, средства проверки достоверности документов. В табл. 7.1 содержится сводная информация, позволяющая получить представление о текущей поддержке спецификации HTML5 Forms.

Таблица 7.1. Поддержка спецификации HTML5 Forms браузерами

Браузер	Описание
Chrome	В версии 5.0 поддерживаются типы элементов ввода `email`, `number`, `tel`, `url`, `search` и `range` и обеспечивается проверка корректности ввода данных
Firefox	В версии 3.6 не поддерживается. Начальная поддержка ожидается в версии 4.0. Неподдерживаемые типы элементов ввода, такие как `url`, `email` и `range`, заменяются текстовыми полями
Internet Explorer	Не поддерживается, но вместо новых типов будут отображаться текстовые поля
Opera	Строгая поддержка первоначальных спецификаций, включая проверку корректности ввода данных
Safari	В версии 4.0 поддерживаются типы элементов ввода `email`, `number`, `tel`, `url`, `search` и `range` и обеспечивается проверка правильности ввода данных. Некоторые типы имеют лучшую поддержку в мобильных версиях Safari

Значимость проверки предоставляемого браузерами уровня поддержки снижается в контексте новых форм HTML5, поскольку в них изначально предусмотрено корректное сокращение возможностей (graceful degradation) в старых браузерах. В целом это означает, что уже сегодня в использовании новых элементов нет никакого риска, поскольку в старых браузерах вместо новых типов элементов ввода, которые они не в состоянии распознать, будут отображаться простые текстовые поля. Однако, как будет показано далее, роль всесторонней проверки достоверности форм при этом только повышается, ибо даже в условиях полной поддержки спецификации HTML5 Forms нельзя быть полностью уверенным в том, что в браузере будет предоставляться валидатор, обеспечивающий соответствие типов данных элементам управления формы.

Каталог элементов ввода

Одним из наилучших источников, в которых можно найти перечень всех новых и измененных элементов HTML5, является список элементов разметки, поддерживаемый на сайте Консорциума W3C. Файл страницы каталога W3C хранится по адресу `http://dev.w3.org/html5/markup`.

На этой странице указаны все существующие и будущие HTML-элементы. В каталоге выделены новые и измененные элементы. Однако слово "новый" в этом списке означает лишь то, что элемент был добавлен после утверждения спецификации HTML4, а не то, что он уже реализован в браузерах или включен в окончательную версию спецификации. Помня об этом уточнении, рассмотрим новые элементы

формы, появившиеся вместе с HTML5, начиная с тех, которые реализуются уже сегодня. В табл. 7.2 перечислены новые значения атрибутов `type`. Многим HTML-разработчикам хорошо знакомы выражения типа `<input type="text">` или `<input type="checkbox">`. Новые типы элементов ввода следуют той же модели, что и существующие.

Таблица 7.2. Новые элементы HTML5 Forms, появляющиеся в браузерах

Браузер	Описание
`tel`	Телефонный номер
`email`	Текстовое поле адреса электронной почты
`url`	URL-адрес в Интернете
`search`	Выражение, предоставляемое поисковой системе. Для этого, например, используется панель поиска в верхней части окна браузера
`range`	Числовой селектор, работающий в определенном интервале значений и обычно визуализируемый в виде ползунка

Что дают эти новые элементы? Если говорить о программировании, то... практически ничего. Действительно, для таких типов, как `tel`, `email`, `url` и `search`, не существует атрибутов, которые отличали бы их от простейшего типа элементов ввода, каковым является тип `text`.

Так в чем же тогда вы выиграете, указав для элемента ввода один из вышеперечисленных специальных типов? Вы получите специализированный элемент управления. (Возможны некоторые ограничения. В случае многих настольных браузеров это ничего не даст.)

Проиллюстрируем сказанное на примере. Указав, что типом элемента ввода является `email` (`<input type="email">`) вместо использования обычного стандарта для текстового поля (`<input type="text">`), вы подсказываете браузеру, что в тех случаях, когда это возможно, он должен представить элемент ввода как-то иначе. Кроме того, вы можете предоставить браузеру возможность выполнить дополнительную проверку корректности ввода данных в этом поле перед их отправкой, но об этом мы поговорим немного позже.

Одними из тех, кто быстрее всего отреагировал на возможность предоставления поддержки новых типов элементов управления, были производители браузеров для мобильных устройств, которые стали использовать различные представления пользовательского интерфейса в зависимости от объявленного типа элемента. Например, стандартная экранная клавиатура, отображаемая в Apple iPhone для элемента ввода типа `text`, имеет вид, представленный на рис. 7.1.

Но если для поля ввода указан тип `email`, то iPhone отображает клавиатуру с другой раскладкой, специально приспособленной для ввода адресов электронной почты, как показано на рис. 7.2.

Обратите внимание на некоторые изменения в раскладке клавиатуры: рядом с клавишей пробела появились новые клавиши, позволяющие вводить символ @ и упрощающие ввод точки (.). Точно так же раскладка клавиатуры претерпевает некоторые изменения для типов `url` и `search`. Однако в настольной версии браузера Safari (и в любом другом браузере, не предоставляющем явной поддержки типов `email`, `url`, `search` и `tel`) будет отображаться лишь обычное текстовое поле. Возможно, в будущих версиях браузеров, даже настольных, будут выводиться визуальные подсказки или символические изображения, указывающие пользователю на то, что поле относится к определенному подтипу. Например, Opera будет отображать небольшое изображение конверта рядом с текстовым полем, указывая на то, что ожидается ввод адреса электронной почты. Однако уже сегодня вы можете без всякого

риска использовать специализированные типы в своих веб-приложениях, поскольку любой браузер либо оптимизирует визуальный вывод применительно к конкретному типу элемента ввода, либо вообще не предпринимает никаких специальных действий.

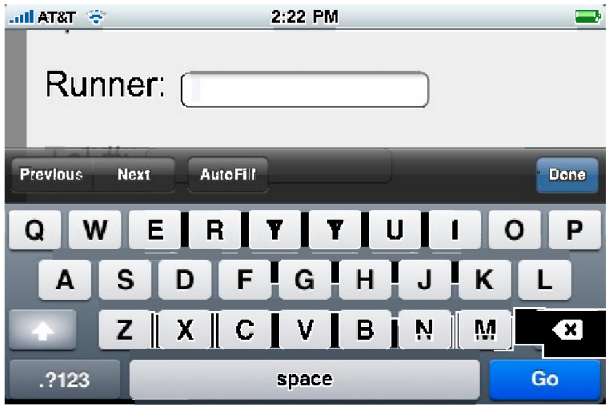


Рис. 7.1. Отображение экранной клавиатуры для элемента ввода типа `text`

Рис. 7.2. Отображение экранной клавиатуры для элемента ввода типа `email`

Еще одним специализированным типом, который начинает интенсивно внедряться в различные браузеры, является тип элементов ввода `<input type="range">`. Этот специализированный элемент управления позволяет выбирать нужное число из заданного интервала значений. Например, этот элемент управления удобно использовать в форме для выбора возраста из диапазона значений, ограниченного снизу, скажем, 18-ю годами. Создав элемент ввода `<input>` типа `<range>` и указав для него значения параметров `min` и `max`, разработчик может потребовать, чтобы страница отображала элемент выбора числовых значений в установленных пределах. Например, в браузере Opera элемент управления

```
<input type="range" min="18" max="120">
```

обеспечивает удобный способ выбора подходящего значения возраста из ограниченного диапазона значений. В самой последней версии браузера Opera этот элемент имеет следующий вид.

К сожалению, тип range сам по себе не обеспечивает отображения числовых значений в браузере. Более того, не имея такой возможности, пользователю достаточно трудно судить о том, какое именно значение выбрано в данный момент. Можно легко исправить этот недостаток, добавив обработчик событий onchange, обновляющий поле индикатора на основании изменений текущего значения элемента ввода типа range, как показано в листинге 7.1.

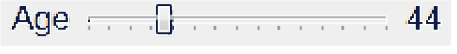
Листинг 7.1. Обработчик событий onchange, обновляющий поле индикации

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

function showValue(newVal) {
  document.getElementById("ageDisplay").innerHTML = newVal;
}
</script>

<label for="age">Age</label>
<input id="age" type="range" min="18" max="120" value="18"
       onchange="showValue(this.value)">
<span id="ageDisplay">18</span>
```

Теперь поле индикатора выглядит более элегантно.



В настоящее время поддержка элементов ввода типа range добавлена в Opera и браузеры на основе WebKit — Safari и Chrome. В Firefox поддержка запланирована, но на момент написания книги точные сроки еще не были определены. Вместо элементов ввода типа range в Firefox пока что будут отображаться простые текстовые элементы.

Здесь водятся драконы!

Говорит Брайан: "Говорят, в былые времена фразой "Здесь водятся драконы!" обозначали места на карте, в которых путешественников могли подстерегать всевозможные опасности. То же самое можно сказать в отношении описанных ниже элементов форм. Хотя спецификации для них написаны и существуют уже довольно длительное время, большинство из них страдает отсутствием фактических реализаций.

Поэтому следует ожидать, что к тому времени, когда разработчики достаточно долго поэкспериментируют с дизайном, сгладят все острые углы, учтут ответную реакцию сообщества и внесут соответствующие правки, многое может измениться. Вместо того чтобы воспринимать компоненты, о которых идет речь ниже, как данность, рассматривайте их как ориентиры, указывающие направление развития форм HTML5. Любые попытки применения их сегодня будут предприниматься вами на свой страх и риск."

В табл. 7.3 приведен перечень дополнительных элементов форм, которые уже запланированы, но пока еще не имеют широкой поддержки.

Хотя некоторые из ранних реализаций этих элементов уже начинают появляться в современных браузерах (например, индикатор даты и времени в Opera, показанный на рис. 7.3), мы не будем задерживаться на них в этой главе, поскольку они, по всей видимости, еще претерпят значительные изменения. Будьте готовы к тому, что впоследствии они могут быть пересмотрены!

Таблица 7.3. Будущие элементы HTML5 Forms

Тип	Назначение
`number`	Поле, содержащее лишь числовые значения
`color`	Селектор цвета, который может быть представлен цветовым кругом или цветовой палитрой
`datetime`	Индикатор даты и времени с учетом часовых поясов (рис. 7.3)
`datetime-local`	Индикатор даты и времени без настройки и учета часовых поясов
`time`	Индикатор и селектор времени без учета часовых поясов
`date`	Селектор календарных дат
`week`	Селектор недель в пределах указанного года
`month`	Селектор месяцев в пределах указанного года

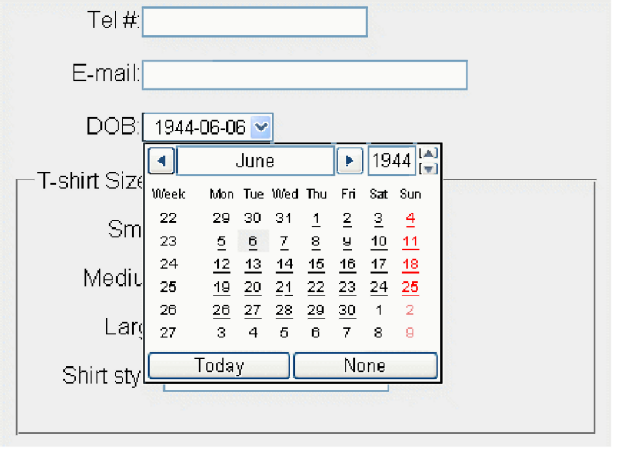


Рис. 7.3. Отображение элемента ввода типа `datetime`

Программный интерфейс HTML5 Forms

Уделив достаточно много времени знакомству с новыми типами элементов форм, вернемся к рассмотрению атрибутов и функций, которые присутствуют как в старых, так и в новых элементах управления форм. Многие из них предназначены для уменьшения объема сценарного кода, необходимого для создания расширенного пользовательского интерфейса веб-приложений. Возможно, эти новые атрибуты дадут вам возможность улучшить пользовательский интерфейс своих приложений способами, о которых вы раньше не задумывались. По крайней мере, это позволит вам убрать некоторые блоки сценариев из уже существующих страниц.

Новые атрибуты и функции форм

Сначала мы рассмотрим новые атрибуты, функции и элементы, которых не было в ранних версиях HTML. Подобно новым типам элементов ввода, можно безопасно использовать эти атрибуты уже сегодня, независимо от того, поддерживает ли их ваш целевой браузер. Это обусловлено тем, что такие атрибуты будут безопасно игнорироваться любым браузером из тех, которые имеются на рынке на сегодняшний день, если браузер не сможет их распознать.

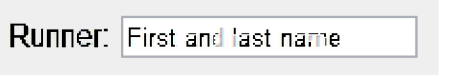
Атрибут `placeholder`

Атрибут `placeholder` позволяет элементам ввода предоставлять замещающий описательный текст, который отображается лишь до тех пор, пока пользователь не введет какое-либо значение. Этот прием довольно распространен во многих современных фреймворках пользовательских интерфейсов, а популярные JavaScript-фреймворки обеспечивают эмуляцию этого средства. Однако в современных браузерах эта возможность является встроенной.

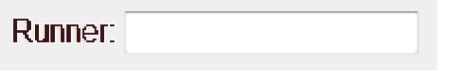
Чтобы использовать этот атрибут, достаточно добавить его в элемент ввода вместе с соответствующим текстом. Применяться он может как с базовым текстовым типом, так и с семантическим типами: `email`, `number`, `url` и т.п.

```
<label>Runner: <input name="name"
 placeholder="First and last name" required></label>
```

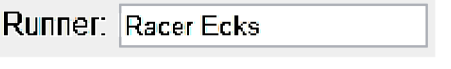
В таких браузерах, как Google Chrome, которые поддерживают данный атрибут, в поле ввода отобразится замещающий текст в приглушенных тонах, который исчезнет, как только пользователь или приложение переведут фокус в это поле или когда в нем уже присутствует какое-либо значение.



Тот же самый атрибут при выполнении в браузере, который его не поддерживает, будет просто игнорироваться, и отображение поля будет соответствовать его поведению, заданному по умолчанию.



Аналогичным образом, если в поле введено значение, замещающий текст отображаться не будет.



Атрибут `autocomplete`

Наконец-то атрибут `autocomplete`, введенный в Internet Explorer 5.5, окончательно стандартизован. Ура! (Браузеры поддерживают этот атрибут практически с момента его появления, но когда поведение атрибута задается спецификацией, то все от этого только выигрывают.)

Атрибут `autocomplete` должен использоваться для защиты критически важных данных пользователя от небезопасного сохранения в локальных файлах браузера. Возможные типы его поведения описаны в табл. 7.4.

Таблица 7.4. Поведение атрибута `autocomplete` в элементах управления вводом

Тип поведения	Назначение
`on`	Поле не защищено, и его значение можно сохранять и извлекать
`off`	Поле защищено, и его значение не должно сохраняться
`unspecified`	По умолчанию принимает значение, установленное для атрибута `autocomplete` элемента `<form>`, содержащего данный элемент. Если же атрибут `autocomplete` не присвоен форме или его значение не задано, то элемент ввода ведет себя так, как если бы для этого атрибута было установлено значение `on`

Атрибут autofocus

С помощью атрибута autofocus разработчик может указать, что данный элемент формы должен получать фокус ввода сразу же после загрузки страницы. На странице должен быть только один атрибут autofocus. Если этот атрибут указан для нескольких элементов, его поведение становится неопределенным.

Примечание

Если ваше содержимое визуализируется на портале или странице с разделяемым содержимым, то проследить за тем, чтобы на странице присутствовал только один атрибут autofocus, очень трудно. Рассчитывать на ожидаемое поведение элементов можно только в том случае, если вы полностью контролируете всю страницу.

Для автоматической передачи фокуса таким элементам управления, как текстовые поля поиска, достаточно задать атрибут autofocus для этого элемента.

```
<input type="search" name="criteria" autofocus>
```

Как и в случае других булевых атрибутов, в указании значения true для атрибута autofocus нет никакой необходимости, поскольку его значение определяется уже самим фактом его присутствия или отсутствия.

Примечание

Автофокусировка может раздражать пользователей, если только за этим не стоит какая-либо очевидная логика. Для навигации по странице многие пользователи используют клавиатуру, и автоматическое переключение фокуса на какой-то один элемент формы может им мешать. Используйте данное средство лишь тогда, когда полностью уверены в том, что форма сможет по-прежнему реагировать на все стандартные клавиатурные нажатия.

Атрибут list и элемент datalist

Комбинация атрибута list и элемента datalist позволяет разработчику указать список возможных значений для ввода. Чтобы воспользоваться этой комбинацией, выполните следующие действия.

1. Создайте в своем документе элемент datalist с уникальным значением атрибута id. Этот элемент может находиться в любом месте документа.
2. Поместите внутри элемента datalist столько элементов option, сколько необходимо для того, чтобы предоставить полный набор возможных значений элемента управления. Например, элемент datalist, представляющий адреса электронной почты контактных лиц, должен содержать их в виде отдельных дочерних элементов option.

```
<datalist id="contactList">
   <option value="x@example.com" label="Racer X">
   <option value="peter@example.com" label="Peter">
</datalist>
```

3. Свяжите элемент ввода с элементом datalist, установив для атрибута list значение, совпадающее со значением идентификатора id соответствующего элемента datalist.

```
<input type="email" id="contacts" list="contactList">
```

В браузерах, поддерживающих эту возможность, например Opera, выводится следующий пользовательский список.

Атрибуты `min` и `max`

Как было продемонстрировано на примере элемента `<input type="range">`, атрибуты `min` и `max` позволяют ограничить допустимый ввод числовых данных минимальным и максимальным значениями. В зависимости от ситуации можно предоставлять один или оба атрибута или вообще их не указывать, и элемент ввода будет соответствующим образом адаптироваться к увеличению или уменьшению интервала допустимых значений. Например, чтобы создать элемент `range`, представляющий степень уверенности участника в собственных силах, выражаемую значениями в интервале от 0 до 100%, можно использовать следующий код:

```
<input id="confidence" name="level" type="range" min="0" max="100"
       value="0">
```

Этот код приведет к созданию элемента управления `range`, для которого минимальным допустимым значением будет `0`, а максимальным — `100`, что в данном случае совпадает со значениями по умолчанию для этого элемента.

Атрибут `step`

Для типов элементов, ожидающих ввода числовых значений, дополнительно предусмотрен атрибут `step`, указывающий величину увеличения или уменьшения значений в процессе регулировки диапазона. Например, рассмотренный выше элемент управления, позволяющий указать степень уверенности участника в своих силах, может быть настроен с помощью атрибута `step` следующим образом:

```
<input id="confidence" name="level" type="range" min="0" max="100"
       step="5" value="0">
```

Таким образом, допустимые значения могут увеличиваться с приращениями в пять единиц, начиная с минимального граничного значения интервала. Иными словами, допустимыми являются лишь значения 0, 5, 10, 15... 100, которые могут вводиться либо с помощью клавиатуры, либо с помощью ползунка, в зависимости от того, как браузер визуализирует данный элемент ввода.

Значение по умолчанию атрибута `step` зависит от типа элемента управления, к которому он применяется. Для элемента ввода типа `range` значением шага по умолчанию является единица. Дополнительно к атрибуту `step` HTML5 вводит для элемента `input` две функции, обеспечивающие управление значениями: `stepUp()` и `stepDown()`.

Как вы, возможно, уже догадались, эти функции позволяют соответственно увеличивать или уменьшать текущее значение. Величина, на которую может быть увеличено или уменьшено значение, совпадает с величиной шага. Таким образом, значение числового элемента ввода можно регулировать без вмешательства пользователя.

Функция `valueAsNumber()`

Новая функция `valueAsNumber()` весьма удобна для преобразования значения элемента управления из текстового формата в числовой... и наоборот! Это возможно благодаря тому, что функция `valueAsNumber()` является одновременно и функцией-получателем значения, и функцией-установщиком. При вызове в качестве функции-получателя она преобразует текстовое значение, находящееся в поле

ввода, в числовую форму, которая может участвовать в вычислениях. Если текстовое значение не может быть корректно преобразовано в тип `number`, возвращается значение `NaN` (Not-a-Number — *не-число*).

Функцию `valueAsNumber()` можно также использовать для установки значения элемента ввода числового типа. Например, для установки диапазона значений степени уверенности можно было бы использовать следующий вызов:

```
document.getElementById("confidence").valueAsNumber(65);
```

Следите за тем, чтобы числовые параметры соответствовали ограничениям, устанавливаемым атрибутами `min`, `max` и `step`, иначе возникнет ошибка.

Атрибут `required`

Если для элемента ввода задан атрибут `required`, то прежде чем форма сможет быть отправлена, этому элементу должно быть присвоено значение. Например, чтобы сделать обязательной установку значения в текстовом поле, достаточно добавить атрибут `required`, как показано ниже.

```
<input type="text" id="firstname" name="first" required>
```

Если значение этого поля не установлено ни программным путем, ни пользователем, то возможность отправки формы блокируется. Атрибут `required` обеспечивает простейшую проверку действительности формы, но возможности проверки формы намного шире. Перейдем к более подробному обсуждению этих возможностей.

Проверка достоверности форм

Прежде чем углубляться в специфику вопроса, рассмотрим, что в целом подразумевается под *валидацией* формы, т.е. проверкой ее действительности (достоверности). По существу, валидация формы — это процесс обнаружения недействительных (невалидных) управляющих данных и донесение информации об обнаруженных ошибках конечному пользователю. Иными словами, валидация формы — это последовательность всевозможных проверок и уведомлений, предоставляющих пользователю возможность корректирования элементов формы, прежде чем она будет отправлена на сервер.

И все-таки, в чем заключена сущность проверки достоверности формы? Ответ простой — в оптимизации.

Проверка достоверности формы является оптимизацией, поскольку сама по себе она еще не гарантирует, что отправляемые на сервер формы синтаксически корректны и содержат данные, отвечающие необходимым требованиям. Этот процесс имеет оптимизационный характер, поскольку предназначен для выявления обстоятельств, при которых приложение может дать сбой, на как можно более ранних стадиях. Иными словами, лучше всего известить пользователя о том, что на странице содержатся недействительные элементы управления формы, непосредственно из самой страницы, используя для этого встроенные в браузер средства обработки документов. Есть ли смысл "гонять" форму по кругу в сети лишь для того, чтобы получить от сервера сообщение, информирующее об опечатках, допущенных при вводе данных? Если только браузер располагает возможностями своевременного вылавливания ошибок, прежде чем форма покинет клиента, мы обязательно должны этим воспользоваться.

Однако самой лишь проверки форм браузером еще недостаточно для обработки всех ошибок.

Злонамеренные действия или неправильное толкование?

Говорит Брайан: "Несмотря на то что в спецификации HTML5 вопросу повышения эффективности проверки форм в браузерах уделено достаточно много внимания, она пока не позволяет отказаться от проверки форм на сервере. Возможно, этого никогда и не произойдет.

Совершенно очевидно, что существует множество обстоятельств, при которых валидация формы требует вмешательства сервера. В качестве примера можно назвать проверку авторизации кредитной карточки при совершении покупки или просто аутентификацию ее владельца. Однако даже при выполнении самых обычных вариантов проверки нельзя полагаться исключительно на клиентскую часть. Некоторые пользователи могут применять браузеры, не поддерживающие средства валидации форм. Часть пользователей может вообще отключить использование сценариев, в результате чего окажутся отключенными все средства проверки форм, кроме самых простейших, основанных на анализе атрибутов. Некоторые пользователи могут использовать такие средства, как подключаемый модуль Firefox Greaseminkey, и изменять содержимое страниц по своему усмотрению. Под этим также может подразумеваться отключение любых видов проверки достоверности форм. В итоге выходит, что полагаться только на клиентскую сторону в качестве единственного средства проверки ввода критически важных данных никак нельзя. Если проверка осуществляется на стороне клиента, ею всегда можно манипулировать.

Средства валидации форм HTML5 обеспечивают быструю обратную связь с пользователями, но ограничиваться только ими не стоит!

В HTML5 предусмотрено восемь удобных способов задания ограничений, обеспечивающих корректность ввода данных в элементы управления. Рассмотрим их по очереди, начав с объекта, предоставляющего информацию о состоянии элементов: `ValidityState`.

В браузерах, поддерживающих средства проверки форм HTML5, возможен доступ к объекту `ValidityState` для любого элемента управления формы.

```
var valCheck = document.myForm.myInput.validity;
```

Данная простая команда получает ссылку на объект `ValidityState` элемента управления формы с красноречивым названием `myInput`. Этот объект предоставляет удобные ссылки на проверочные поля, каждое из которых содержит результат проверки одного из восьми доступных ограничений (условий), а также на итоговый результат такой проверки для данного элемента.

С помощью вызова `valCheck.valid` мы можем получить булево значение, позволяющее судить о том, удовлетворяет или не удовлетворяет данный элемент управления формы требуемым ограничениям. Считайте, что флаг `valid` играет роль индикатора итогового результата проверки: если проверка выполнения всех восьми ограничений оказалась успешной, то этот флаг примет значение `true`. В противном случае, если нарушается хотя бы одно из условий, определяющих действительность значения элемента, значением флага будет `false`.

Примечание

Объект `ValidityState` — активный. После того как ссылка на него получена, к нему можно в любой момент обратиться, и возвращенные им результаты проверки будут соответствовать текущему состоянию элемента.

Как уже отмечалось ранее, для любого заданного элемента формы существует восемь проверяемых ограничений. Доступ к результату проверки каждого из них осуществляется через поле объекта `ValidityState` с соответствующим именем. Рассмотрим, что означают эти ограничения, как проверить их выполнение для данного элемента управления формы и как использовать объект `ValidityState` для получения их значений.

Ограничение `valueMissing`

Назначение. Обеспечивает проверку установки значения для данного элемента управления формы.

Использование. Установите для данного элемента формы атрибут `required`.

Пример использования:

```
<input type="text" name="myText" required>
```

Описание. Если для элемента управления формы установлен атрибут `required`, то результат проверки будет положительным только в том случае, если пользователь или программный вызов установили для данного элемента некоторое значение. Например, пустое текстовое поле не пройдет эту проверку, но оно пройдет ее сразу же после того, как в него будет введен любой текст. Для пустого поля проверка `valueMissing` вернет значение `true`.

Ограничение `typeMismatch`

Назначение. Обеспечивает проверку того, что тип введенного значения совпадает с ожидаемым (`number`, `email`, `URL` и т.п.).

Использование. Установите для элемента управления одно из подходящих значений атрибута `type`.

Пример использования:

```
<input type="email" name="myEmail">
```

Описание. Специальные типы элементов управления формы существуют не только для того, чтобы отображать в мобильных телефонах нестандартные виртуальные клавиатуры! Если ваш браузер в состоянии определить, что введенное значение не согласуется с правилами, установленными для элемента данного типа, — например, адрес электронной почты введен без символа `@`, — то браузер отметит этот элемент флажком, указывающим на несоответствие введенного значения заданному типу. В качестве другого примера можно привести поле типа `number`, данные в котором не удается интерпретировать как число. В любом из этих случаев проверка `typeMismatch` вернет `true`.

Ограничение `patternMismatch`

Назначение. Обеспечивает проверку того, что значение элемента управления формы соответствует шаблону, описывающему допустимые форматы данных.

Использование. Установите для элемента управления формы атрибут `pattern`, указав для него соответствующий шаблон.

Пример использования:

```
<input type="text" name="creditcardnumber" pattern="[0-9]{16}"
title="A credit card number is 16 digits with no spaces or dashes">
```

Описание. Атрибут `pattern` предоставляет в распоряжение разработчика мощный и гибкий способ задания шаблона регулярных выражений для значений элемента управления формы. Если значение элемента, для которого задан шаблон, не соответствует этому шаблону, проверка `patternMismatch` вернет `true`. Чтобы предоставить дополнительную информацию, предназначенную для пользователей и специальных утилит, в любом поле, контролируемом шаблонами, можно установить атрибут `title`, содержащий описание правил форматирования.

Ограничение `tooLong`

Назначение. Обеспечивает ограничение количества символов, используемых в значении.

Использование. Поместите в элемент управления формы атрибут maxLength.

Пример использования:

```
<input type="text" name="limitedText" maxlength="140">
```

Описание. Это ограничение возвращает true, если длина значения превышает maxlength. В то время как длина значений, вводимых пользователем в поля формы, обычно контролируется уже на этапе ввода, в некоторых ситуациях, включая установку значений программным путем, их длина может выходить за установленные пределы.

Ограничение rangeUnderflow

Назначение. Обеспечивает задание минимального допустимого значения для элемента управления формы.

Использование. Установите для элемента управления формы атрибут min с соответствующим значением.

Пример использования:

```
<input type="range" name="ageCheck" min="18">
```

Описание. Для любого элемента управления формы, в котором используется проверка выхода числовых значений за границы диапазона, можно временно разрешить использование значений, величина которых меньше нижней допустимой границы. В подобных случаях проверка ValidityState вернет для поля rangeUnderflow значение true.

Ограничение rangeOverflow

Назначение. Обеспечивает задание максимального допустимого значения для элемента управления формы.

Использование. Установите для элемента управления формы атрибут max с соответствующим значением.

Пример использования:

```
<input type="range" name="kidAgeCheck" max="12">
```

Описание. По аналогии с родственным ограничением rangeUnderflow, возвращает true, если значение элемента управления формы превышает значение атрибута max.

Ограничение stepMismatch

Назначение. Гарантирует, что значение будет удовлетворять условиям, определяемым комбинацией атрибутов min, max и step.

Использование. Установите для атрибута step значение, задающее допустимую величину приращения.

Пример использования:

```
<input type="range" name="confidenceLevel" min="0" max="100"
       step="5">
```

Описание. Это ограничение обеспечивает соблюдение условий, определяемых комбинацией атрибутов min, max и step. В частности, текущее значение должно представлять собой сумму минимального значения и величины, кратной значению атрибута step. Например, в случае диапазона значений от 0 до 100 с шагом 5 значение 17 не будет допущено без того, чтобы проверка stepMismatch не вернула значение true.

Ограничение customError

Назначение. Соответствует обработке ошибок, явно вычисляемых и устанавливаемых самим приложением.

Использование. Для приведения элемента управления формы в состояние customError вызовите метод setCustomValidity(message).

Пример использования:

```
passwordConfirmationField.setCustomValidity(" Пароль не совпадает.")
```

Описание. В ситуациях, когда встроенные методы проверки оказываются неприменимыми, может пригодиться нестандартная обработка ошибок. Для тех случаев, когда значение, введенное в поле, не согласуется с семантическими правилами, приложение должно задать соответствующие сообщения.

Типичным примером ситуаций, в которых возникает необходимость в использовании нестандартной проверки, является несогласованность значений управляющих элементов формы между собой, что, в частности, может происходить, когда значение пароля, введенное при его подтверждении, не совпадает с первоначально введенным значением. (Более подробно этот пример рассматривается в конце главы.) Во всех подобных случаях недействительности элемента управления формы возвращаемое для него значение customError будет равно true. Чтобы сбросить ошибку, достаточно вызвать для данного элемента метод setCustomValidity("") с пустой строкой в качестве аргумента.

Проверочные поля и функции

Совокупность восьми описанных выше проверочных полей позволяет разработчику точно выяснить, почему тот или иной элемент управления формы не смог пройти проверку. Если же конкретные причины неудачного исхода проверки вас не интересуют, ограничьтесь получением булевого свойства valid объекта validityState, которое агрегирует в себе результаты проверки выполнения всех восьми ограничений. Если для всех восьми проверочных полей возвращается значение false, то поле valid вернет значение true. Существует несколько других полезных полей и методов элементов управления формы, облегчающих выполнение программной проверки действительности элементов.

Атрибут willValidate

Атрибут willValidate просто указывает на то, будет ли данный элемент управления формы вообще подвергаться проверке. Если для данного элемента управления установлено хотя бы одно из ограничений, — например, с помощью атрибутов required, pattern и т.д., — то поле willValidate сообщит вам, что проверка будет производиться.

Функция checkValidity()

Функция checkValidity() позволяет проверить действительность формы без явного выполнения пользовательского ввода. Обычно проверка формы осуществляется в момент ее отправки пользователем или сценарием. С помощью данной функции такую проверку можно выполнить в любое время.

Примечание

Вызов метода checkValidity() из элемента управления формы приводит не только к выполнению проверки корректности элемента, но и к запуску тех же результирующих событий и триггеров, что и при отправке формы.

Атрибут `validationMessage`

Этот атрибут пока что не поддерживается текущими версиями браузеров, но к тому времени, когда вы будете читать эти строки, ситуация может измениться. С помощью атрибута `validationMessage` можно программно запрашивать сообщения, которые браузер будет отображать, если элемент по тем или иным причинам не пройдет проверку достоверности. Например, если в поле, для которого задан атрибут `required`, не введено значение, то браузер мог бы выводить для пользователя следующее сообщение об ошибке: "This field requires a value" ("В это поле должно быть введено значение"). Именно эту текстовую строку и возвратит поле `validationMessage`, если оно поддерживается, причем конкретный текст возвращаемого сообщения будет изменяться в зависимости от текущего состояния результата проверки данного элемента управления.

Обратная связь с пользователем при проверке ввода значений в элементы формы

Обсудим теперь вопрос об обратной связи с пользователем при проверке значений, введенных в форму. Этот вопрос касается того, когда и каким образом браузер должен предоставить пользователю возможность вмешаться для устранения ошибок. Спецификация не оговаривает, как именно должен обновляться пользовательский интерфейс для представления сообщений об ошибках, и в этом отношении существующие реализации значительно различаются между собой. Рассмотрим, например, как это реализовано в браузере Opera. В Opera 10.5 ошибочное поле окрашивается в красный цвет и начинает мигать, а рядом с ним выводится всплывающее сообщение с соответствующим текстом.

В отличие от этого, браузер Google Chrome при обнаружении ошибки лишь переходит к соответствующему полю и помещает в него фокус ввода. Какое поведение следует считать правильным?

Об этом в спецификации не говорится ни слова. Однако, если у вас возникнет желание взять на себя управление обратной связью с пользователем в случае обнаружения неверно введенных данных, в вашем распоряжении имеется предназначенное для этого средство: событие `invalid`.

Всякий раз, когда осуществляется проверка достоверности формы, — будь-то при отправке формы или в результате непосредственного вызова функции `checkValidity()`, — недействительной форме будет доставлено событие `invalid`. Это событие можно проигнорировать, принять его во внимание и даже отменить. Чтобы добавить обработчик событий к элементу, который будет получать это уведомление, используйте код наподобие того, который представлен в листинге 7.2.

Листинг 7.2. Добавление обработчика событий `invalid`

```
// Обработчик событий invalid
function invalidHandler(evt) {
  var validity = evt.srcElement.validity;
```

```
  // Проверить соблюдение некоторого ограничения
  if (validity.valueMissing) {
    // Уведомить пользователя о том, что в поле не введено значение
  }

  // Возможная проверка соблюдения других ограничений...

  // Если хотите предотвратить вывод браузером стандартных сообщений
  // об ошибках при проверке формы, отмените событие
  evt.preventDefault();
}

// Зарегистрировать обработчик событий invalid
myField.addEventListener("invalid", invalidHandler, false);
```

Рассмотрим этот код по частям.

Сначала мы объявляем обработчик событий `invalid`. В этом обработчике мы прежде всего выясняем источник события. Вспомните, что событие `invalid` запускается элементом, который не проходит проверку достоверности. Таковым в этой функции является элемент `evt.srcElement`.

Зная источник ошибки, мы получаем объект `validity`. Используя этот экземпляр объекта `ValidityState`, мы можем проверить значения полей отдельных ограничений для определения конкретных причин возникновения ошибки. В данном случае, поскольку мы знаем, что наше поле содержит атрибут `required`, целесообразно сначала проверить состояние поля `valueMissing`, указывающее на возможный пропуск входных данных.

Если окажется, что данные действительно не были введены, мы можем модифицировать пользовательский интерфейс на странице таким образом, чтобы информировать пользователя о необходимости ввести значение в поле, вызвавшее ошибку. Решение о том, что именно следует отобразить — окно предупреждений или сообщение об ошибке в отдельной информационной области страницы, вы принимаете по собственному усмотрению.

После того как пользователь извещен о возникновении ошибки и снабжен инструкциями относительно того, как ее исправить, необходимо принять решение относительно того, следует ли предоставить браузеру возможность использовать собственные средства обратной связи с пользователем. По умолчанию именно так браузер и поступает. Чтобы предотвратить вывод собственных сообщений браузера, можно воспользоваться вызовом `evt.preventDefault()`, что позволит отменить обработку ошибок, предусмотренную по умолчанию, и позаботиться об этом самостоятельно.

Следует вновь подчеркнуть, что выбор остается полностью за вами. Спецификация HTML5 Forms предоставляет вам значительную свободу действий в отношении того, воспользоваться специализированными функциями или ограничиться возможностями браузерами, используемыми по умолчанию.

Отключение проверки достоверности формы

Несмотря на широкие возможности, предлагаемые функциями проверки достоверности форм, возможны случаи, когда желательно отключить такую проверку для какого-либо элемента управления или формы в целом. Чаще всего необходимость в этом может возникать в связи с тем, что вы хотите отправить форму с целью сохранения ее временного содержимого или его последующего извлечения, даже если оно пока не является полностью достоверным.

Представьте ситуацию, когда пользователю, заполняющему сложную форму заказа, потребовалось срочно отлучиться с места работы для выполнения какого-либо поручения. В идеальном случае можно было использовать кнопку Сохранить, которая позволила бы сохранить уже введенные данные путем отправки формы на сервер. Но если форма заполнена лишь частично, то правила проверки достоверности могут сделать отправку формы невозможной. Таким образом, если работа пользователя прерывается в результате возникновения непредвиденных обстоятельств, он должен либо завершить заполнение формы, либо проделанная часть работы окажется напрасно выполненной.

Чтобы иметь возможность справляться с подобными ситуациями, можно установить для самой формы атрибут `noValidate`, в результате чего форма успешно обойдет любую предусмотренную логику проверки достоверности и будет отправлена на сервер. Разумеется, этот атрибут формы может быть установлен как сценарием, так и исходной разметкой.

Более полезный способ отключения проверки состоит в том, чтобы установить атрибут `formNoValidate` для таких элементов, как кнопка отправки формы (кнопки `submit`). Ниже приведен пример кода, который заставляет кнопку `submit` играть роль кнопки сохранения формы.

```
<input type="submit" formnovalidate name="save"
       value="Save current progress">
<input type="submit" name="process" value="Process order">
```

В этом фрагменте кода создаются две кнопки `submit`. Вторая кнопка выполняет функции обычной кнопки отправки формы. В то же время первой кнопке присвоен атрибут `formNoValidate`, что приведет к обходу всех проверок при использовании этой кнопки. Благодаря этому данные можно отправлять на сервер без проверки их корректности. Конечно же, сервер должен быть настроен для обработки непроверенных данных, однако практика показывает, что поступать так следует всегда, независимо от обстоятельств.

Создание приложения на основе формы HTML5

Теперь настало время воспользоваться описанными в этой главе средствами для создания простой страницы регистрации, демонстрирующей новые возможности форм HTML5. Возвратимся к нашему старому знакомому приложению Happy Trails Running Club и создадим страницу регистрации участников состязаний, включающую в себя новые элементы формы и средства проверки их достоверности.

Как обычно, приведенный ниже код демонстрационных файлов доступен в папке `code/forms`. Стилям CSS и периферийной разметке мы уделим меньше внимания, сосредоточившись главным образом на основной части самой страницы. Мы начнем с общего вида страницы, представленного на рис. 7.4, а затем разобьем ее на части.

Эта страница демонстрирует использование многих элементов и функций, изученных нами ранее, включая средства проверки достоверности элементов формы. Хотя фактическая картинка в вашем браузере может несколько отличаться от приведенной, она, в соответствии с принципом корректного сокращения возможностей, должна отобразиться даже в том случае, если какое-то средство браузером не поддерживается.

Рис. 7.4. Пример страницы регистрации участников соревнований

Итак, приступим к анализу кода. Начальная, навигационная и завершающая части страницы уже рассматривались нами в предыдущих примерах. Теперь страница дополнительно содержит элемент `<form>`.

```
<form name="register">
  <p><label for="runnername">Runner:</label>
     <input id="runnername" name="runnername" type="text"
    placeholder="First and last name" required></p>
  <p><label for="phone">Tel #:</label>
     <input id="phone" name="phone" type="tel"
    placeholder="(xxx) xxx-xxx"></p>
  <p><label for="emailaddress">E-mail:</label>
     <input id="emailaddress" name="emailaddress" type="email"
    placeholder="For confirmation only"></p>
  <p><label for="dob">DOB:</label>
```

```
    <input id="dob" name="dob" type="date"
   placeholder="MM/DD/YYYY"></p>
```

В этом фрагменте мы видим разметку для четырех основных элементов ввода, с помощью которых участник может ввести свои данные: имя, номер телефона, адрес электронной почты и дату дня рождения. Для каждого из них мы задали элемент `<label>`, содержащий описательный текст, и связали его с самим элементом ввода с помощью атрибута `for`. Кроме того, мы указали для каждого элемента ввода замещающий текст, информирующий пользователя о типе содержимого данного поля.

Текстовое поле, предназначенное для ввода имени участника забега (`runnername`), сделано обязательным для ввода путем указания для него атрибута `required`. Если в этом поле ничего не будет введено, то в процессе проверки достоверности формы сработает ограничение `valueMissing`. В качестве типа поля для ввода телефонного номера (`phone`) указано значение `tel`. В зависимости от возможностей вашего браузера это поле может отображаться по-разному или предоставлять оптимизированные варианты виртуальной клавиатуры.

Аналогичным образом, для поля адреса электронной почты (`emailaddress`) указан тип `email`. Специфика его обработки зависит от возможностей браузера. Если введенное значение не соответствует правильному формату, то в некоторых браузерах сработает ограничение `typeMismatch`.

Наконец, для поля даты рождения объявлен тип `date`. Пока что этот тип поддерживают немногие браузеры, но если он поддерживается, то рядом с этим полем автоматически отобразится элемент управления, обеспечивающий выбор даты.

```
<fieldset>
  <legend>T-shirt Size: </legend>
  <p><input id="small" type="radio" name="tshirt" value="small">
     <label for="small">Small</label></p>
  <p><input id="medium" type="radio" name="tshirt" value="medium">
     <label for="medium">Medium</label></p>
  <p><input id="large" type="radio" name="tshirt" value="large">
     <label for="large">Large</label></p>
  <p><label for="style">Shirt style:</label>
     <input id="style" name="style" type="text" list="stylelist"
            title="Years of participation"></p>
  <datalist id="stylelist">
   <option value="White" label="1st Year">
   <option value="Gray" label="2nd - 4th Year">
   <option value="Navy" label="Veteran (5+ Years)">
  </datalist>
</fieldset>
```

В этом фрагменте кода устанавливаются элементы управления, которые используются для выбора характеристик футболки участника (T-shirt). Первую группу этих элементов образует набор переключателей, предназначенных для выбора размера футболки.

Следующий фрагмент кода гораздо интереснее предыдущего. Здесь мы используем атрибут `<list>` и соответствующий ему элемент `<datalist>`. В элементе `<datalist>` объявляется набор опций, которые должны отображаться в этом списке с различными значениями атрибутов `value` и `label` и представлять имеющиеся типы футболок в зависимости от спортивного стажа участника соревнований. Этот список довольно простой, но тот же самый прием можно использовать и в случае длинных списков динамических элементов.

```
<fieldset>
   <legend>Expectations:</legend>
```

```
    <p>
    <label for="confidence">Confidence:</label>
    <input id="confidence" name="level" type="range"
           onchange="setConfidence(this.value)"
           min="0" max="100" step="5" value="0">
    <span id="confidenceDisplay">0%</span></p>
    <p><label for="notes">Notes:</label>
       <textarea id="notes" name="notes" maxLength="140">
       </textarea></p>
</fieldset>
```

В последнем из фрагментов кода, описывающих элементы управления, создается ползунок, с помощью которого участник может указать степень своей уверенности в том, что он сможет пробежать всю дистанцию. Для этой цели используется элемент ввода типа `range`. Поскольку интересующая нас величина измеряется в процентах, мы устанавливаем для элемента ввода атрибуты `min`, `max` и `step`. Тем самым задается обычный для процентных величин диапазон допустимых значений. Кроме того, значение шага изменения задано равным 5%, что можно будет визуально наблюдать, если этот элемент управления поддерживается браузером. В этом элементе можно использовать ограничения проверки достоверности `rangeUnderFlow`, `rangeOverFlow` и `stepMismatch`, хотя при обычном взаимодействии пользователя с данным ползунком эти ограничения не могут быть нарушены.

Поскольку текстовое представление значений элемента управления `range` по умолчанию не предусмотрено, мы должны добавить его самостоятельно. Для манипулирования значениями элемента `span` с идентификатором `confidenceDisplay` используется обработчик событий `onchange` элемента управления `range`, и вскоре мы увидим, как это работает.

Наконец, мы добавляем элемент `<textarea>`, в который могут быть помещены любые дополнительные сведения о регистрирующемся участнике соревнований. Установив для него атрибут `maxlength`, мы заставляем этот элемент подчиняться ограничению `tooLong`, что пресечет попытки ввода в это поле слишком длинных значений.

```
  <p><input type="submit" name="register" value="Register"></p>
</form>
```

Раздел, в котором создаются элементы управления формы, заканчивается кнопкой `submit` с надписью **Register**, которая отвечает за отправку регистрационной формы. В нашем иллюстративном примере данная кнопка является фиктивной, и никакой фактической отправки регистрационных данных на сервер не происходит.

Остается сказать несколько слов о сценариях, которые используются для обновления показаний индикатора ползунка и подмены встроенных средств браузера, обеспечивающих обратную связь с пользователем при проверке достоверности формы. Хотя во многих случаях средства обработки ошибок при вводе данных формы, предусмотренные в браузере по умолчанию, могут оказаться вполне приемлемыми, знание дополнительных возможностей, которые имеются в вашем распоряжении, никогда не помешает.

```
<script type="text/javascript">
    function setConfidence(newVal) {
        document.getElementById("confidenceDisplay").innerHTML =
            newVal + '%';
    }

    function invalidHandler(evt) {
        // Найти элемент label, соответствующий данному элементу
        // управления формы
        var label = evt.srcElement.parentElement.
```

```
                        getElementsByTagName("label")[0];

        // Установить красный цвет шрифта для этого элемента
        label.style.color = 'red';

        // Предотвратить дальнейшее распространение события
        evt.stopPropagation();

        // Отменить стандартную обработку ошибок браузером при
        // проверке достоверности
        evt.preventDefault();
    }

    function loadDemo() {
       // Зарегистрировать для формы обработчик событий, который
       // будет обрабатывать все уведомления об ошибках ввода
       // данных в форму
       document.register.addEventListener("invalid",
                                          invalidHandler, true);
    }

    window.addEventListener("load", loadDemo, false);

</script>
```

Этот код состоит из нескольких частей. Первая из них представляет функцию `setConfidence()`. Она предназначена для обработки событий изменений, происходящих в элементе управления `range`. При каждом изменении положения ползунка находится элемент `span` с идентификатором `confidenceDisplay` и соответствующим образом изменяется его текст. Как можно увидеть, все это легко конфигурируется.

Примечание

Почему для элементов `range` визуальная индикация не предусмотрена по умолчанию? Возможно, это сделано потому, что проектировщики пользовательского интерфейса могут точнее рассчитать положение и внешний вид индикатора применительно к конкретным обстоятельствам. Использование визуальной индикации в качестве дополнительной возможности добавляет немного работы, но зато повышает гибкость.

Оставшаяся часть кода показывает, как подменить стандартную обработку ошибок, возникающих при проверке достоверности формы. Мы начинаем с регистрации обработчика для специального типа событий — `invalid`. Чтобы перехватывать события `invalid` от всех элементов управления формы, мы регистрируем обработчик для формы в целом.

```
// Зарегистрировать для формы обработчик событий, который будет
// обрабатывать все уведомления об ошибках ввода данных в форму
document.register.addEventListener("invalid", invalidHandler, true)
```

Теперь всякий раз, когда для какого-либо элемента управления формы сработает ограничение проверки достоверности, будет вызываться обработчик событий `invalidHandler()`. Чтобы предоставить более элегантный вариант обратной связи с пользователем, чем те, которые по умолчанию предлагаются некоторыми известными браузерами, выделим поле, вызвавшее ошибку, красным цветом. Для этого сначала найдем элемент `label` путем перехода к родительскому элементу.

```
// Найти элемент label, соответствующий данному элементу
// управления формы
var label = evt.srcElement.parentElement.
                getElementsByTagName("label")[0];
```

```
// Установить красный цвет шрифта для этого элемента
label.style.color = 'red';
```

Далее мы предотвращаем повторную обработку события `invalid` браузером или любым другим обработчиком. С этой целью воспользуемся мощными возможностями DOM, вызвав метод `stopPropagation()`, чтобы воспрепятствовать перехвату события другими обработчиками, и метод `preventDefault()`, чтобы отменить обработку события браузером, предусмотренную по умолчанию.

```
// Предотвратить дальнейшее распространение события
evt.stopPropagation();

// Отменить стандартную обработку ошибок браузером при проверке
// достоверности
evt.preventDefault();
```

Вот так, с помощью всего лишь нескольких простых действий, нам удалось создать собственный специализированный интерфейс для взаимодействия со средствами проверки достоверности формы!

Дополнительные рекомендации

Некоторые методики, не нашедшие отражения в основных примерах, приведенных в данной книге, могут быть весьма полезными во многих типах HTML5-приложений. Ниже представлены небольшие фрагменты кода, представляющие практический интерес, вместе с соответствующими комментариями и рекомендациями.

Проверка пароля

С помощью средств проверки достоверности форм HTML5 можно реализовать верификацию паролей при их смене. Стандартная методика предусматривает предоставление двух полей для ввода пароля. При этом обязательным условием успешной отправки формы является совпадение значений пароля, введенных в обоих полях. Ниже показано, как осуществить проверку совпадения обоих значений перед отправкой формы, используя функцию `setCustomValidity()`.

Вспомните, что ограничение `customError` предоставляет возможность обнаруживать ошибки в элементах управления формы в случаях, не охватываемых стандартными правилами. В частности, удобная возможность для использования ограничения `customError` предоставляется в тех случаях, когда достоверность формы зависит от согласованности состояний нескольких элементов управления, что как раз и требуется в случае двух вышеупомянутых полей пароля.

Поскольку предполагается, что объект `validityState` остается активным, как только получена ссылка на него, целесообразно устанавливать для этого объекта состояние нестандартной ошибки всякий раз, когда значения пароля, введенные в обоих полях, не совпадают между собой, и сбрасывать это состояние сразу же после того, как только совпадение содержимого в этих полях восстановится. Чтобы реализовать эту идею, воспользуемся обработчиком событий `onchange`, связанным с полями пароля.

```
<form name="passwordChange">
    <p><label for="password1">Новый пароль:</label>
    <input type="password" id="password1"
           onchange="checkPasswords()"></p>
    <p><label for="password2">Подтвердите пароль:</label>
    <input type="password" id="password2"
           onchange="checkPasswords()"></p>
</form>
```

В рассматриваемом простом случае формы с двумя полями для ввода пароля мы можем зарегистрировать функцию, которая будет выполняться при всяком изменении любого из значений пароля.

```
function checkPasswords() {
  var pass1 = document.getElementById("password1");
  var pass2 = document.getElementById("password2");

  if (pass1.value != pass2.value)
    pass1.setCustomValidity("Пароли не совпадают. Пожалуйста,
                             проверьте идентичность паролей
                             в обоих полях!");
  else
    pass1.setCustomValidity("");
}
```

Для проверки совпадения паролей мы просто извлекаем значения пароля, введенные в обоих полях, и в случае их несовпадения между собой устанавливаем нестандартную ошибку. Вероятно, вполне приемлемо было бы устанавливать состояние ошибки только для одного из полей. Если же оба значения одинаковы, ошибка сбрасывается путем установки для нее значения в виде пустой строки. Такой способ сброса нестандартных ошибок оговорен в спецификации.

После того как для поля установлена ошибка, можно, используя описанные в этой главе методы, отобразить интерфейс обратной связи с пользователем, предоставив ему возможность правильно ввести пароль.

Резюме

В этой главе было показано, как, исходя из привычных объектов — HTML-форм, — получить нечто новое путем использования новых элементов, атрибутов и функций, доступных в HTML5. Вы познакомились с несколькими новыми типами элементов ввода, но их должно появиться еще больше. Вы увидели, как клиентские средства проверки достоверности формы могут интегрироваться непосредственно в элементы управления формы с целью предотвращения бесполезной пересылки неправильных данных. Было показано, как уменьшить объем кода сценариев, используемых для создания в приложениях полноценного пользовательского интерфейса. В следующей главе описываются возможности браузеров, связанные с порождением нескольких независимых потоков выполнения для трудоемких задач.

Глава 8

Технология Web Workers

Технология HTML5 Web Workers предоставляет веб-приложениям возможности фоновой обработки в виде отдельных потоков выполнения, что позволяет JavaScript-приложениям в полной мере использовать преимущества многоядерных процессоров. Кроме того, разделение трудоемких задач между несколькими потоками Web Workers позволяет избежать появления назойливых предупреждений о слишком долго выполняющихся сценариях (рис. 8.1), которые отображаются, если выполнение циклов JavaScript длится дольше нескольких секунд.

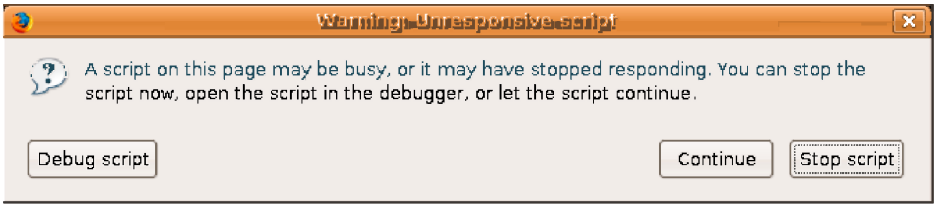


Рис. 8.1. Предупреждение относительно длительно выполняющегося сценария JavaScript в браузере Firefox

Какими бы мощными возможностями ни обладали потоки выполнения Web Workers, существуют некоторые вещи, которые нельзя сделать с их помощью. Например, если сценарий выполняется внутри потока, то он не имеет доступа к объекту `window` веб-страницы (`window.document`), а это означает, что у потоков отсутствует возможность непосредственного доступа к веб-странице и DOM. Хотя потоки не могут блокировать пользовательский интерфейс браузера, они все же потребляют ресурсы процессора и замедляют работу системы.

Предположим, вам необходимо создать веб-приложение, выполняющее интенсивные фоновые вычисления, но вы не хотите, чтобы это затрудняло интерактивное взаимодействие с самой веб-страницей. Используя технологию Web Workers, можно породить поток, предназначенный для выполнения указанных вычислений, и добавить перехватчик событий, который будет перехватывать сообщения потока по мере их отправки.

В качестве другого типичного примера использования потоков Web Workers можно привести приложение, которое получает широковещательные сообщения с сервера новостей и размещает их на основной веб-странице по мере их получения с сервера. Для связи с сервером новостей такие потоки могут использовать вебсокеты или серверные события.

В этой главе изучаются возможности технологии Web Workers. Сначала мы рассмотрим принципы работы потоков Web Workers и уровень поддержки этой технологии в браузерах, доступный на момент написания книги. Затем мы обсудим использование API-функций для создания новых потоков и способы организации связи между потоком и породившим его контекстом. Наконец, будет показано, как построить приложение, использующее технологию Web Workers.

Поддержка спецификации HTML5 Web Workers браузерами

Как видно из табл. 8.1, на момент написания книги спецификация HTML5 Web Workers уже поддерживалась многими браузерами.

Таблица 8.1. Поддержка спецификации HTML5 Web Workers браузерами

Браузер	Описание
Chrome	Поддерживается в версии 3 и выше
Firefox	Поддерживается в версии 3.5 и выше
Internet Explorer	Не поддерживается (в настоящее время)
Opera	Поддерживается в версии 10.6 и выше
Safari	Поддерживается в версии 4 и выше

Программный интерфейс HTML5 Web Workers

В этом разделе подробно рассматривается использование программного интерфейса Web Workers. Для иллюстрации мы создали простую страницу в браузере: `echo.html`. Использование технологии Web Workers не составляет большого труда — вы создаете объект `Worker` и передаете ему файл JavaScript, подлежащий выполнению. Далее следует настроить обработчики событий для перехвата сообщений и кодов ошибок, посылаемых потоком Web Workers, а если требуется связь страницы с потоком, следует вызвать метод `postMessage()` для передачи данных. То же самое относится к коду JavaScript-файла в потоке: следует настроить обработчики событий для обработки поступающих сообщений и кодов ошибок, а для осуществления связи со страницей — использовать метод `postMessage()`.

Проверка поддержки в браузере

Прежде чем использовать программный интерфейс Web Workers, следует убедиться, что браузер предоставляет необходимую поддержку. Если такая поддержка отсутствует, вы сможете предоставить замещающий текст, предлагающий пользователям приложения установить более современный браузер. Один из возможных способов тестирования поддержки в браузере демонстрируется в листинге 8.1.

Листинг 8.1. Проверка поддержки в браузере

```
function loadDemo() {
  if (typeof(Worker) !== "undefined") {
    document.getElementById("support").innerHTML =
            "Прекрасно! Ваш браузер поддерживает HTML5 Web Workers";
  }
}
```

В этом примере браузерная поддержка тестируется с помощью функции `loadDemo()`, которая может вызываться при загрузке страницы. Вызов `typeof(Worker)` вернет глобальное свойство окна `Worker`, которое в случае отсутствия поддержки программного интерфейса Web Workers в браузере будет иметь значение `undefined`. В этом примере страница сообщает о наличии или отсутствии необходимой поддержки в браузере путем замены содержимого ранее определенного на странице элемента `support` соответствующим сообщением, как показано на рис. 8.2.

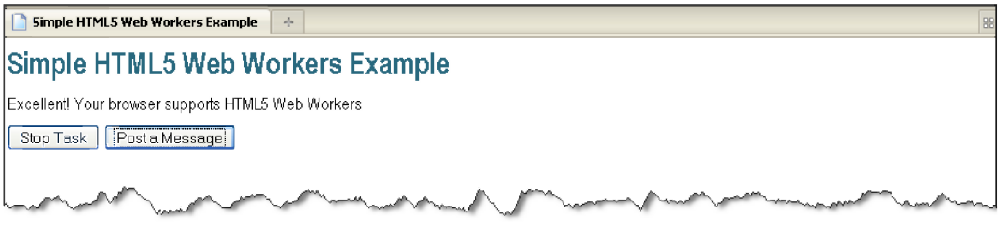


Рис. 8.2. Пример, иллюстрирующий наличие поддержки HTML5 Web Workers в браузере

Создание потоков Web Workers

Потоки Web Workers инициализируются URL-адресом JavaScript-файла, в котором содержится код потока, подлежащего выполнению. Этот код устанавливает обработчик событий и обменивается сообщениями с породившим его сценарием. URL-адрес JavaScript-файла может задаваться в виде относительного или абсолютного адреса с тем же источником (т.е. схемой, хостом и портом), что и основная страница.

```
worker = new Worker("echoWorker.js");
```

Загрузка и выполнение дополнительного JavaScript-кода

Приложение, в состав которого входит несколько JavaScript-файлов, может содержать элементы `<script>`, осуществляющие синхронную загрузку JavaScript-файлов по мере загрузки страницы. Но поскольку потоки Web Workers не имеют доступа к объекту `document`, мы используем другой возможный механизм, обеспечивающий синхронное импортирование JavaScript-файлов изнутри потоков: функцию `importScripts()`.

```
importScripts("helper.js");
```

Импортируемый JavaScript-файл просто загружается в существующий поток и выполняется в нем. Одним вызовом `importScripts()` можно импортировать сразу несколько сценариев. В этом случае сценарии выполняются в той очередности, в которой они указаны:

```
importScripts("helper.js", "anotherHelper.js");
```

Обмен сообщениями с потоками

После того как поток создан, с ним можно обмениваться сообщениями, используя для этого метод `postMessage()`. Это тот же метод `postMessage()`, который используется для обмена сообщениями между фреймами и окнами. С помощью данного метода можно пересылать большинство JavaScript-объектов, за исключением функций и объектов с циклическими ссылками.

Предположим, вы решили создать простое приложение с потоками Web Workers, которым пользователи могли бы отправлять сообщения и которые, в свою очередь, могли бы отвечать пользователям путем обратной отправки этих же сообщений. Возможно, на практике от этого приложения будет мало толку, но оно достаточно полезно для иллюстрации решений, которые потребуются для создания более сложных примеров.

Пример веб-страницы такого приложения и его потока представлен на рис. 8.3. Полный код этой простой страницы приведен в конце раздела.

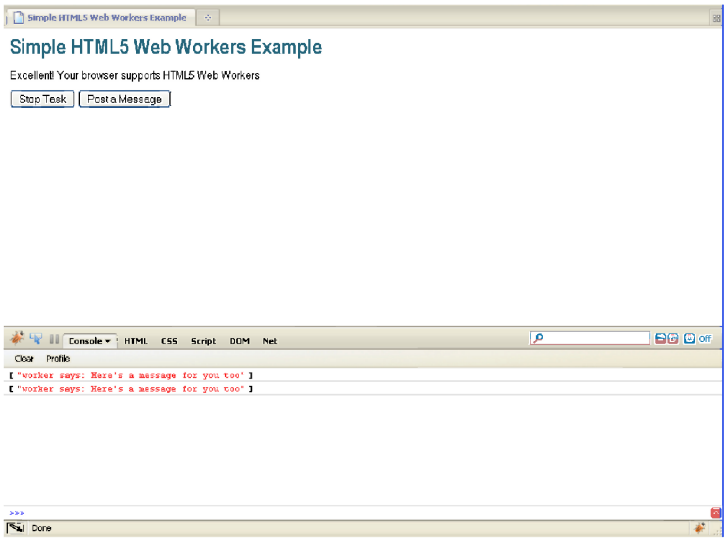


Рис. 8.3. Простая веб-страница, в которой используется поток Web Workers

Чтобы организовать связь с потоком, потребуется добавить некоторый код на *основную страницу* (страница, которая вызывает поток выполнения) и в *JavaScript-файл рабочего потока*.

Добавление кода на основную страницу

Чтобы направлять сообщения из страницы в поток Web Workers, должен вызываться метод `postMessage()`, который осуществляет передачу необходимых данных. Для получения сообщений и кодов ошибок, отправляемых из потока в страницу, следует установить обработчик событий.

Чтобы организовать обмен сообщениями между основной страницей и потоком, прежде всего следует добавить вызов метода `postMessage()` на основную страницу, как показано ниже.

```
document.getElementById("helloButton").onclick = function() {
    worker.postMessage("Вот еще одно сообщение для вас");
}
```

В этом примере сообщение направляется потоку при выполнении щелчка на кнопке **Отправить сообщение** (Post a Message). Далее следует добавить на страницу обработчик, перехватывающий сообщения, направляемые потоком.

```
worker.addEventListener("message", messageHandler, true);

function messageHandler(e) {
    // Обработать сообщение, отправленное потоком
}
```

Добавление кода в JavaScript-файл потока

Теперь необходимо добавить аналогичный код в JavaScript-файл потока. В этом коде необходимо установить обработчики событий для обработки поступающих сообщений и кодов ошибок, тогда как отправка сообщений странице будет осуществляться путем вызова метода `postMessage()`.

Прежде всего, для завершения цикла обмена сообщениями между страницей и потоком добавьте вызов метода `postMessage()`; это может быть сделано, например, внутри функции `messageHandler()`.

```
function messageHandler(e) {
    postMessage("Поток уведомляет: " + e.data);
}
```

Затем добавьте обработчик событий в JavaScript-файл потока, который обрабатывает сообщения, поступающие из основной страницы.

```
addEventListener("message", messageHandler, true);
```

Здесь функция `messageHandler()` вызывается сразу же после получения сообщения, так что оно может быть немедленно отправлено обратно.

Обработка ошибок

Необработанные ошибки в сценарии потока запускают события `error` в объекте `Worker`. Перехват этих событий особенно важен при отладке сценариев, использующих потоки Web Workers. Ниже показан пример функции обработки ошибок в JavaScript-файле потока. Эта функция выводит сообщения об ошибках на консоль.

```
function errorHandler(e) {
    console.log(e.message, e);
}
```

Для обработки ошибок необходимо добавить на основную страницу обработчик событий.

```
worker.addEventListener("error", errorHandler, true);
```

Прекращение выполнения потоков

Потоки Web Workers не могут самостоятельно прекратить работу; остановить их может лишь страница, которая их запустила. Если в потоке больше нет необходимости (например, если основная страница получила уведомление о том, что поток завершил выполнение предписанных ему действий), следует освободить занимаемые им ресурсы. Кроме того, в некоторых случаях, если поток выполняется слишком долго, пользователь может захотеть преждевременно прекратить его выполнение. Чтобы прекратить выполнение потока, следует вызвать метод `terminate()`. Поток, выполнение которого прекращено, перестает отвечать на сообщения и выполнять любые другие вычисления. Перезапустить поток нельзя; вместо этого следует создать новый поток, использовав для этого тот же URL.

```
worker.terminate();
```

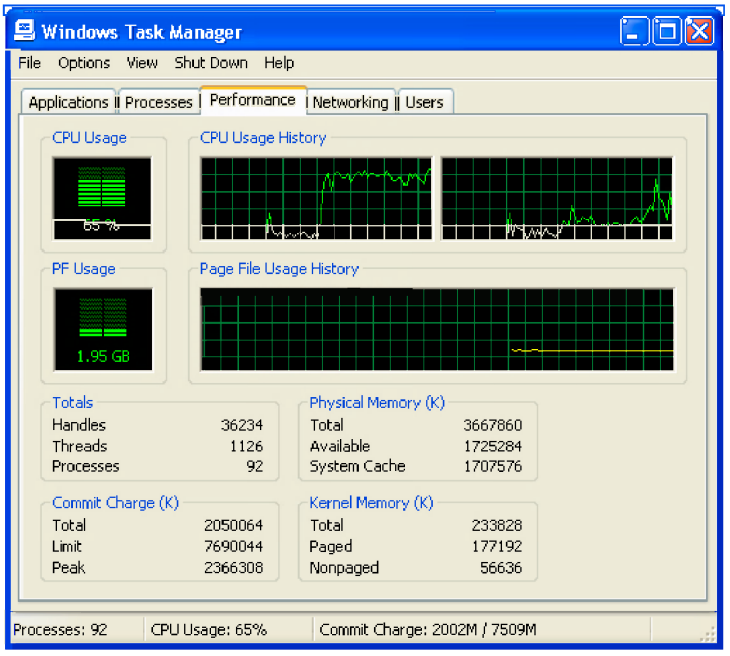
Использование одних потоков внутри других

В сценариях, запускающих потоки Web Workers, можно создавать подчиненные потоки с помощью объекта `Worker`.

```
var subWorker = new Worker("subWorker.js");
```

Экспериментируйте с параллельным выполнением нескольких потоков

Говорит Питер: "Породив поток, рекурсивно порождающий другой поток с тем же самым исходным JavaScript-файлом, вы сможете сделать ряд весьма интересных наблюдений."



Использование таймеров

Несмотря на то что потоки Web Workers не имеют доступа к объекту `window`, они могут использовать весь набор функций JavaScript, предназначенных для работы со временем, которые обычно предоставляются глобальным объектом окна.

```
var t = setTimeout(postMessage, 2000, "delayed message");
```

Простой пример

Для полноты изложения в листингах 8.2 и 8.3 приводится полный код простой страницы (файл `code/workers/echo.html`) и JavaScript-файла для работы с потоками Web Workers (файл `code/workers/echoWorkers.js`).

Листинг 8.2. Простая HTML-страница, создающая новый поток выполнения

```
<!DOCTYPE html>
<title>Простой пример потоков HTML5 Web Workers</title>

<link rel="stylesheet" href="styles.css">

<h1>Простой пример потоков HTML5 Web Workers</h1>

<p id="support">Ваш браузер не поддерживает HTML5 Web Workers</p>

<button id="stopButton" >Остановить задачу</button>
<button id="helloButton" >Отправить сообщение</button>


<script>
   function stopWorker() {
```

```
        worker.terminate();
    }

    function messageHandler(e) {
        console.log(e.data);
    }

    function errorHandler(e) {
        console.warn(e.message, e);
    }

    function loadDemo() {
      if (typeof(Worker) !== "undefined") {
          document.getElementById("support").innerHTML =
            "Прекрасно! Ваш браузер поддерживает HTML5 Web Workers";

          worker = new Worker("echoWorker.js");
          worker.addEventListener("message", messageHandler, true);
          worker.addEventListener("error", errorHandler, true);

          document.getElementById("helloButton").onclick =
            function() {
              worker.postMessage("Вот еще одно сообщение для вас");
            }

          document.getElementById("stopButton").onclick = stopWorker;
      }
    }

window.addEventListener("load", loadDemo, true);

</script>
```

Листинг 8.3. Простой JavaScript-файл для работы с потоком

```
function messageHandler(e) {
    postMessage("Поток уведомляет: " + e.data);
}

addEventListener("message", messageHandler, true);
```

Создание приложения на основе технологии HTML5 Web Workers

До сих пор наше внимание было сосредоточено на использовании различных функций Web Worker API. Попробуем теперь оценить истинные возможности потоков выполнения Web Workers, создав полноценное приложение: веб-страницу, использующую фильтр размытия изображений, выполнение которой распараллелено между несколькими потоками. Работу этого приложения иллюстрирует рис. 8.4.

Приложение рассылает данные изображения из элемента `canvas` нескольким потокам Web Workers (количество потоков можно задавать). Далее изображение обрабатывается потоками с помощью простого фильтра Box Blur (кубическое размытие). Это может занимать несколько секунд, в зависимости от размера изображения и доступных вычислительных ресурсов (даже компьютеры с быстрыми процессами могут быть загружены другими задачами, что приводит к увеличению времени выполнения JavaScript-кода).

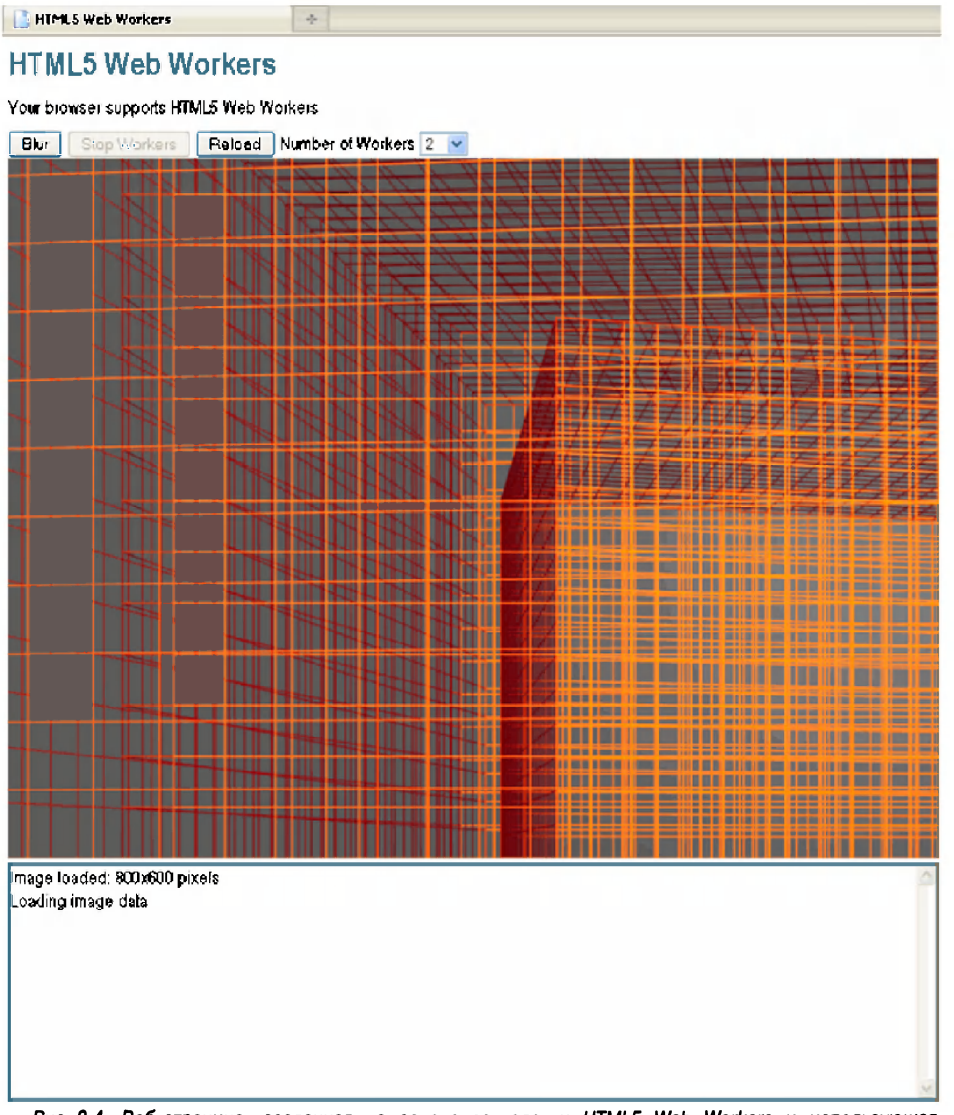


Рис. 8.4. Веб-страница, созданная на основе технологии HTML5 Web Workers и использующая фильтр размытия изображений

Вместе с тем, поскольку интенсивные вычисления переложены на потоки выполнения Web Workers, в данном приложении отсутствует опасность получения предупреждений о затянувшейся работе сценария, а значит, и отпадает необходимость разбивать задачу на отдельные части, вручную выделяя для них определенные промежутки времени, что пришлось бы делать, если бы не использовались потоки.

Код вспомогательного сценария `blur.js`

В файле `blur.js` мы можем использовать непосредственную реализацию фильтра размытия, который выполняется в цикле до тех пор, пока не будут обработаны все входные данные, как показано в листинге 8.4.

Листинг 8.4. Реализация фильтра кубического размытия в файле `blur.js`

```
function inRange(i, width, height) {
    return ((i>=0) && (i < width*height*4));
}
function averageNeighbors(imageData, width, height, i) {
    var v = imageData[i];
    // Главные направления
    var north = inRange(i-width*4, width, height) ?
                imageData[i-width*4] : v;
    var south = inRange(i+width*4, width, height) ?
                imageData[i+width*4] : v;
    var west = inRange(i-4, width, height) ? imageData[i-4] : v;
    var east = inRange(i+4, width, height) ? imageData[i+4] : v;
    // Диагональные направления
    var ne = inRange(i-width*4+4, width, height) ?
             imageData[i-width*4+4] : v;
    var nw = inRange(i-width*4-4, width, height) ?
             imageData[i-width*4-4] : v;
    var se = inRange(i+width*4+4, width, height) ?
             imageData[i+width*4+4] : v;
    var sw = inRange(i+width*4-4, width, height) ?
             imageData[i+width*4-4] : v;
    // Среднее
    var newVal = Math.floor((north + south + east + west + se +
                             sw + ne + nw + v)/9);
    if (isNaN(newVal)) {
        sendStatus("bad value " + i + " for height " + height);
        throw new Error("NaN");
    }
    return newVal;
}
function boxBlur(imageData, width, height) {
    var data = [];
    var val = 0;
    for (var i=0; i<width*height*4; i++) {
        val = averageNeighbors(imageData, width, height, i);
        data[i] = val;
    }
    return data;
}
```

Если говорить коротко, то данный алгоритм размытия основан на усреднении значений близлежащих пикселей. В случае крупных изображений, насчитывающих миллионы пикселей, для этого может потребоваться значительное время. Запуск подобных циклов в интерфейсных потоках крайне нежелателен. Даже если предупреждение о чрезмерно длительном выполнении сценария и не появится, пользовательский интерфейс страницы сможет отвечать на интерактивные запросы пользователя только после того, как завершится выполнение цикла. Поэтому выполнение вычислений в фоновых потоках Web Workers служит хорошим примером.

Код страницы приложения `blur.html`

В листинге 8.5 приведен код HTML-страницы, которая запускает потоки выполнения. HTML-код для этого примера выбран достаточно простым, чтобы все в ней было понятно. Нашей целью является не создание привлекательного интерфейса,

а предоставление простого каркаса, который позволяет управлять потоками Web Workers и демонстрирует их применение. В этом приложении в страницу вставляется элемент canvas, который отображает входное изображение. Предусмотрены кнопки, с помощью которых пользователь может запускать и останавливать процесс размытия изображения, а также сбрасывать изображение. Кроме того, предоставляется возможность указать количество используемых потоков.

Листинг 8.5. Код страницы `blur.html`

```
<!DOCTYPE html>
<title>HTML5 Web Workers</title>
<link rel="stylesheet" href = "styles.css">

<h1>HTML5 Web Workers</h1>

<p id="status">Ваш браузер не поддерживает HTML5 Web Workers</p>


<button id="startBlurButton" disabled>Размыть</button>
<button id="stopButton" disabled>Остановить потоки</button>
<button onclick="document.location = document.location;">
  Перезагрузить</button>

<label for="workerCount">Количество потоков</label>
<select id="workerCount">
    <option>1</option>
    <option selected>2</option>
    <option>4</option>
    <option>8</option>
    <option>16</option>
</select>

<div id="imageContainer"></div>
<div id="logOutput"></div>
```

Добавим в файл `blur.html` код, создающий потоки. Для этого мы создаем экземпляр объекта `Worker`, которому передаем URL-адрес JavaScript-файла. Все экземпляры объекта `Worker` будут выполнять один и тот же код, но обрабатывать различные участки входного ихзображения.

```
function initWorker(src) {
    var worker = new Worker(src);
    worker.addEventListener("message", messageHandler, true);
    worker.addEventListener("error", errorHandler, true);
    return worker;
}
```

Нам остается добавить в файл `blur.html` код, обрабатывающий ошибки. При возникновении ошибки в потоке выполнения страница сможет отобразить сообщение об ошибке. В нашем примере никаких проблем возникать не должно, но перехват событий ошибок обычно является одним из элементов хорошей практики программирования, достоинства которого в процессе отладки трудно переоценить.

```
function errorHandler(e) {
    log("Ошибка: " + e.message);
}
```

Код сценария `blueWorker.js`

Далее мы добавляем в файл `blueWorker.js` код, с помощью которого потоки выполнения смогут обмениваться сообщениями со страницей (листинг 8.6). Когда каждый из потоков заканчивает блок вычислений, он может информировать страницу о проделанной работе с помощью функции `postMessage()`. Мы используем эту информацию для обновления изображения, которое выводится на основной странице. Сразу после создания каждый из потоков ожидает получения сообщения, содержащего данные изображения и команду начать размытие. Это сообщение представляет собой объект JavaScript, содержащий тип сообщения и данные изображения, представленные в виде числового массива.

Листинг 8.6. Отправка и обработка данных изображения в файле `blurWorker.js`

```
function sendStatus(statusText) {
    postMessage({"type" : "status",
                 "statusText" : statusText}
                );
}

function messageHandler(e) {
   var messageType = e.data.type;
   switch (messageType) {
      case ("blur"):
         sendStatus("Потоком начато размытие данных в диапазоне: " +
                    e.data.startX + "-" +
                    (e.data.startX+e.data.width));
         var imageData = e.data.imageData;
         imageData = boxBlur(imageData, e.data.width, e.data.height,
                             e.data.startX);

         postMessage({"type" : "progress",
                      "imageData" : imageData,
                      "width" : e.data.width,
                      "height" : e.data.height,
                      "startX" : e.data.startX
                     });
         sendStatus("Завершено размытие данных в диапазоне: " +
                    e.data.startX + "-" +
                    (e.data.width+e.data.startX));
         break;
      default:
         sendStatus("Потоком получено сообщение: " + e.data);
   }
}
addEventListener("message", messageHandler, true);
```

Передача сообщений потокам

Мы можем использовать потоки в файле `blur.html`, посылая им некоторые данные и аргументы с инструкциями, касающимися размытия изображения. Это обеспечивается использованием метода `postMessage()` для отправки JavaScript-объекта, содержащего массив RGBA-значений, размеры исходного изображения и диапазон пикселей, за размытие которого отвечает данный поток. Каждый из потоков обрабатывает свой участок изображения, инструкции относительно которого он получает вместе с сообщением.

```
function sendBlurTask(worker, i, chunkWidth) {
        var chunkHeight = image.height;
        var chunkStartX = i * chunkWidth;
        var chunkStartY = 0;
        var data = ctx.getImageData(chunkStartX, chunkStartY,
                                    chunkWidth, chunkHeight).data;

        worker.postMessage({'type' : 'blur',
                            'imageData' : data,
                            'width' : chunkWidth,
                            'height' : chunkHeight,
                            'startX' : chunkStartX});
}
```

Данные изображения, хранящегося в элементе `canvas`

Говорит Фрэнк: "Метод `postMessage()` призван обеспечивать эффективную сериализацию объектов `imageData` при использовании совместно с объектом Canvas. Некоторые браузеры, включающие объект `Worker` и метод `postMessage()`, могут не поддерживать расширенные возможности метода. Например, Firefox 3.5 не может пересылать объекты `imageData` с помощью метода `postMessage()`, хотя в будущих версиях Firefox такая возможность может быть предоставлена.

В силу указанных причин, в примере обработки изображений, приведенном в этой главе, пересылаются данные `image.Data.data` (сериализуемые как массив JavaScript), а не сам объект `image.Data`. По мере выполнения потоками Web Workers своих задач они направляют странице сведения о своем статусе и результаты работы. В листинге 8.6 показано, каким образом потоки отправляют данные странице после их обработки фильтром размытия. Как и до этого, сообщение содержит JavaScript-объект с полями для данных изображения и координат обработанного участка."

Что касается HTML-страницы, то обработчик сообщений получает эти данные и использует их для обновления холста новыми значениями пикселей. Результат немедленно отображается на экране по мере поступления данных обработанного изображения. Итак, нам удалось создать пример приложения, способного обрабатывать изображения и использовать возможности многоядерных процессоров. Более того, в процессе выполнения потоков Web Workers пользовательский интерфейс не блокируется и его способность к отклику не ухудшается. Работу приложения иллюстрирует рис. 8.5.

Приложение в действии

Чтобы увидеть пример в действии, необходимо запросить страницу `blur.html` у веб-сервера (например, Apache или Python SimpleHTTPServer). Ниже описаны действия, которые следует выполнить для запуска приложения в случае использования сервера Python SimpleHTTPServer.

1. Установите среду разработки Python.
2. Перейдите в каталог, содержащий файл примера (`blur.html`).
3. Запустите Python с помощью следующей команды:
   ```
   python -m SimpleHTTPServer 9999
   ```
4. Откройте браузер и перейдите на страницу `http://localhost:9999/blur.html`.

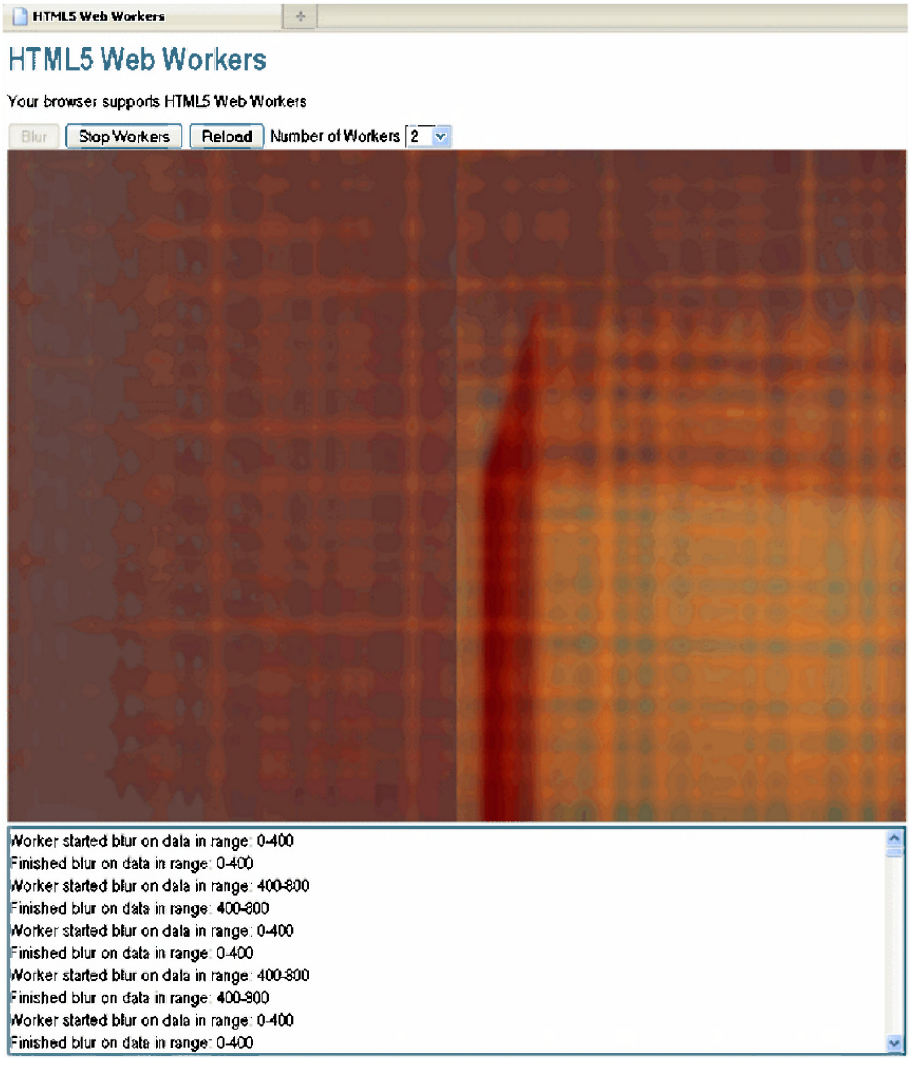


Рис. 8.5. Многопотоковое приложение, выполняющее размытие изображения

Код примера

В листингах 8.7, 8.8 и 8.9 приводится полный код приложения.

Листинг 8.7. Содержимое файла `blur.html`

```
<!DOCTYPE html>
<title>HTML5 Web Workers</title>
<link rel="stylesheet" href = "styles.css">

<h1>HTML5 Web Workers</h1>

<p id="status">Ваш браузер не поддерживает HTML5 Web Workers</p>
```

```
<button id="startBlurButton" disabled>Размыть</button>
<button id="stopButton" disabled>Остановить потоки</button>
<button onclick="document.location = document.location;">
Перезагрузить</button>

<label for="workerCount">Количество потоков</label>
<select id="workerCount">
    <option>1</option>
    <option selected>2</option>
    <option>4</option>
    <option>8</option>
    <option>16</option>
</select>

<div id="imageContainer"></div>
<div id="logOutput"></div>
<script>

var imageURL = "example2.png";
var image;
var ctx;
var workers = [];

function log(s) {
    var logOutput = document.getElementById("logOutput");
    logOutput.innerHTML = s + "<br>" + logOutput.innerHTML;
}

function setRunningState(p) {
    // В процессе выполнения кнопка "stop" активизирована,
    // а кнопка "start" отключена
    document.getElementById("startBlurButton").disabled = p;
    document.getElementById("stopButton").disabled = !p;
}

function initWorker(src) {
    var worker = new Worker(src);
    worker.addEventListener("message", messageHandler, true);
    worker.addEventListener("error", errorHandler, true);
    return worker;
}

function startBlur() {
    var workerCount =
           parseInt(document.getElementById("workerCount").value);
    var width = image.width/workerCount;

    for (var i=0; i<workerCount; i++) {
        var worker = initWorker("blurWorker.js");
        worker.index = i;
        worker.width = width;
        workers[i] = worker;

        sendBlurTask(worker, i, width);
    }
    setRunningState(true);
}
```

```
function sendBlurTask(worker, i, chunkWidth) {
        var chunkHeight = image.height;
        var chunkStartX = i * chunkWidth;
        var chunkStartY = 0;
        var data = ctx.getImageData(chunkStartX, chunkStartY,
                                    chunkWidth, chunkHeight).data;

        worker.postMessage({'type' : 'blur',
                            'imageData' : data,
                            'width' : chunkWidth,
                            'height' : chunkHeight,
                            'startX' : chunkStartX});
}

function stopBlur() {
    for (var i=0; i<workers.length; i++) {
        workers[i].terminate();
    }
    setRunningState(false);
}

function messageHandler(e) {
    var messageType = e.data.type;
    switch (messageType) {
        case ("status"):
            log(e.data.statusText);
            break;
        case ("progress"):
            var imageData = ctx.createImageData(e.data.width,
                                                e.data.height);

            for (var i = 0; i<imageData.data.length; i++) {
                var val = e.data.imageData[i];
                if (val === null || val > 255 || val < 0) {
                    log("illegal value: " + val + " at " + i);
                    return;
                }

                imageData.data[i] = val;
            }
            ctx.putImageData(imageData, e.data.startX, 0);

            // Вновь размыть ту же область
            sendBlurTask(e.target, e.target.index, e.target.width);
            break;
        default:
            break;
    }
}

function errorHandler(e) {
    log("error: " + e.message);
}

function loadImageData(url) {

    var canvas = document.createElement('canvas');
```

```
    ctx = canvas.getContext('2d');
    image = new Image();
    image.src = url;

    document.getElementById("imageContainer").appendChild(canvas);

    image.onload = function(){
        canvas.width = image.width;
        canvas.height = image.height;
        ctx.drawImage(image, 0, 0);
        window.imgdata = ctx.getImageData(0, 0, image.width,
                                          image.height);
        n = ctx.createImageData(image.width, image.height);
        setRunningState(false);
        log("Загружаемое изображение: " + image.width + "x" +
            image.height + " пикселей");
    };
}

function loadDemo() {
   log("Загрузка данных изображения");

   if (typeof(Worker) !== "undefined") {
      document.getElementById("status").innerHTML =
         "Ваш браузер поддерживает HTML5 Web Workers";

      document.getElementById("stopButton").onclick = stopBlur;
      document.getElementById("startBlurButton").onclick = startBlur;

      loadImageData(imageURL);

      document.getElementById("startBlurButton").disabled = true;
      document.getElementById("stopButton").disabled = true;
   }

}

window.addEventListener("load", loadDemo, true);
</script>
```

Листинг 8.8. Содержимое файла `blurWorker.js`

```
importScripts("blur.js");

function sendStatus(statusText) {
    postMessage({"type" : "status",
                 "statusText" : statusText}
                );
}

function messageHandler(e) {
    var messageType = e.data.type;
    switch (messageType) {
        case ("blur"):
            sendStatus("Потоком начато размытие данных в диапазоне: "
+
                       e.data.startX + "-" +
                       (e.data.startX+e.data.width));
```

```
            var imageData = e.data.imageData;
            imageData = boxBlur(imageData, e.data.width,
                                e.data.height, e.data.startX);

            postMessage({"type" : "progress",
                         "imageData" : imageData,
                         "width" : e.data.width,
                         "height" : e.data.height,
                         "startX" : e.data.startX
                        });
            sendStatus("Завершено размытие данных в диапазоне: " +
                       e.data.startX + "-" +
                       (e.data.width+e.data.startX));
            break;
        default:
            sendStatus("Потоком получено сообщение: " + e.data);
    }
}

addEventListener("message", messageHandler, true);
```

Листинг 8.9. Содержимое файла `blur.js`

```
function inRange(i, width, height) {
    return ((i>=0) && (i < width*height*4));
}

function averageNeighbors(imageData, width, height, i) {
    var v = imageData[i];

    // Главные направления
    var north = inRange(i-width*4, width, height) ?
                imageData[i-width*4] : v;
    var south = inRange(i+width*4, width, height)
                ? imageData[i+width*4] : v;
    var west = inRange(i-4, width, height) ? imageData[i-4] : v;
    var east = inRange(i+4, width, height) ? imageData[i+4] : v;

    // Диагональные направления
    var ne = inRange(i-width*4+4, width, height) ?
             imageData[i-width*4+4] : v;
    var nw = inRange(i-width*4-4, width, height) ?
             imageData[i-width*4-4] : v;
    var se = inRange(i+width*4+4, width, height) ?
             imageData[i+width*4+4] : v;
    var sw = inRange(i+width*4-4, width, height) ?
             imageData[i+width*4-4] : v;

    // Среднее
    var newVal = Math.floor((north + south + east + west + se +
                             sw + ne + nw + v)/9);

    if (isNaN(newVal)) {
        sendStatus("bad value " + i + " for height " + height);
        throw new Error("NaN");
    }
    return newVal;
}
```

```
function boxBlur(imageData, width, height) {
    var data = [];
    var val = 0;
    for (var i=0; i<width*height*4; i++) {
        val = averageNeighbors(imageData, width, height, i);
        data[i] = val;
    }
    return data;
}
```

Резюме

В этой главе вы узнали о том, как применять потоки выполнения Web Workers для создания приложений, использующих фоновую обработку данных. Мы обсудили принципы работы потоков Web Workers и уровень их поддержки различными браузерами на момент написания книги. Было показано, как использовать эту технологию для создания новых потоков и как создать приложение, использующее возможности потоков. В следующей главе будут продемонстрированы дополнительные возможности, предлагаемые HTML5 для сохранения локальных копий данных и снижения объема служебного сетевого трафика в приложениях.

Глава 9

Технология Web Storage

В этой главе рассматриваются возможности спецификации HTML5 Web Storage, известной также под названием DomStorage, — программного интерфейса, упрощающего сохранение данных посредством веб-запросов. Раньше удаленные веб-серверы были вынуждены хранить постоянные данные на самом сервере, непрерывно пересылая их между сервером и клиентами. С появлением Web Storage API разработчики получили возможность сохранять данные непосредственно на стороне клиента, в браузере, где они остаются доступными для повторных запросов и могут быть извлечены даже после закрытия браузера, тем самым снижая сетевой трафик.

В этой главе мы сначала познакомимся с отличиями Web Storage от технологии "cookie", а затем изучим различные способы сохранения и извлечения данных. Далее будут рассмотрены отличия локальных хранилищ от хранилищ сеансов, атрибуты и функции, предлагаемые интерфейсом хранилища, а также способы обработки событий Web Storage. Глава завершается рассмотрением технологии Web SQL Database и некоторыми практическими примерами.

Обзор технологии HTML5 Web Storage

Знакомство с технологией Web Storage лучше всего начать с рассмотрения ее предшественника с интригующим названием *cookie*. Сохраняемые в браузере файлы "cookie", унаследовавшие свое название от давней методики программирования *magic cookie*, с помощью которой обеспечивался обмен небольшими порциями данных между программами, являются встроенным механизмом пересылки текстовых значений между сервером и клиентом. Сохраняемые в файлах "cookie" данные могут использоваться серверами для запоминания информации о конкретных пользователях, посещающих веб-страницы на данном сервере. Файлы "cookie" передаются в обоих направлениях при каждом посещении домена пользователем. Например, в файле "cookie" может храниться идентификатор сеанса, по которому веб-сервер может определить, какая именно виртуальная "корзина" принадлежит данному пользователю, сверяя идентификатор с собственной базой данных. По мере перехода пользователя от страницы к странице информация о нем непрерывно обновляется. В качестве другого примера можно привести сохранение в браузере локальных данных приложения, которые могут использоваться при последующих загрузках страницы.

Файлы "cookie" используются серверами и в других целях, не столь приветствуемых пользователями, например для отслеживания сайтов, наиболее часто посещаемых пользователями. Эта информация служит для рассылки рекламы соответствующего содержания. В связи с этим часть пользователей потребовала, чтобы производители браузеров предусмотрѐли возможность блокирования или удаления файлов "cookie" для всех без исключения сайтов или только для некоторых из них.

Нравится это или нет, но файлы "cookie" поддерживаются еще со времен первого выпуска браузера Netscape, а это случилось в середине 90-х годов прошлого столетия. Кроме того, файлы "cookie" относятся к тем немногочисленным средствам,

которые поддерживаются производителями различных браузеров с самых первых дней существования Интернета. Они позволяют хранить данные в промежутках между запросами; для этого требуется лишь тщательно продумать обмен данными между сервером и кодом в браузере. Несмотря на их универсальность, файлам "cookie" свойственны хорошо известные недостатки.

- Размер файлов "cookie" жестко ограничен. Обычно в таком файле может быть передано не более 4 Кбайт данных, что, например, совершенно неприемлемо для передачи документов или сообщений электронной почты.
- Файлы "cookie" передаются от браузера к серверу и обратно при каждом запросе, связанном с конкретным файлом. Это означает, что содержащиеся в файле данные не только видимы в сети, что превращает их в фактор риска, если они не зашифрованы, но и занимают часть полосы пропускания при каждой загрузке страницы. А раз так, то даже сравнительно небольшой размер файлов "cookie" обретает совершенно иное значение.

Во многих случаях нужного результата можно добиться без использования сети или удаленного сервера. Именно здесь на сцену выходит технология HTML5 Web Storage. С помощью простого программного интерфейса разработчики могут сохранять данные в легко извлекаемых JavaScript-объектах, постоянно существующих от одной загрузки страницы к другой. Используя хранилище сеанса или локальное хранилище, разработчики могут сами определять, должны ли данные сохраняться по отдельности для страниц, открытых в одном и том же окне (вкладке), и прекращать существование после закрытия этого окна (вкладки) или же совместно использоваться всеми окнами (вкладками) и продолжать существовать даже после перезапуска браузера. Кроме того, технология Web Storage значительно снижает пороговый объем сохраняемых данных, который, в зависимости от реализации браузера, может составлять несколько мегабайт.

Благодаря этому технология Web Storage оказывается весьма удобной для сохранения документов и файловых данных, которые могут быстро разрастаться в размерах и превышать пределы, установленные для файлов "cookie".

Поддержка спецификации HTML5 Web Storage браузерами

Web Storage относится к числу наиболее широко внедряемых спецификаций HTML5. По существу, все версии браузеров, поставляемые в настоящее время, в той или иной мере поддерживают Web Storage (табл. 9.1).

Таблица 9.1. Поддержка спецификации HTML5 Web Storage браузерами

Браузер	Описание
Chrome	Поддерживается в версии 3.0 и выше
Firefox	Поддерживается в версии 3.0 и выше
Internet Explorer	Поддерживается в версии 8.0 и выше
Opera	Поддерживается в версии 10.5 и выше
Safari	Поддерживается в версии 4.0 и выше

Учитывая столь широкую поддержку этой спецификации, HTML5 Web Storage является на сегодняшний день одним из самых безопасных программных интерфейсов, используемых в веб-приложениях. Но все же, перед тем как использовать Web Storage, целесообразно сначала убедиться в наличии необходимой поддержки

в браузере. О том, как выполнить такую проверку программным путем, будет рассказано далее.

Программный интерфейс HTML5 Web Storage

Программный интерфейс Web Storage на удивление прост в использовании. Сначала мы рассмотрим процедуры сохранения и извлечения значений, а затем перейдем к обсуждению различий между локальными и сеансовыми хранилищами. После этого мы изучим более сложные аспекты данной спецификации, такие как событийные уведомления об изменении значений.

Проверка поддержки в браузере

Доступ к базе данных хранилища для данного домена осуществляется непосредственно из объекта `window`. Поэтому определение того, поддерживает ли браузер пользователя интерфейс Web Storage, сводится к проверке существования объекта `window.sessionStorage` или `window.localStorage`. В листинге 9.1 приведена подпрограмма, которая проверяет, поддерживается ли хранилище, и отображает сообщение о том, поддерживает ли данный браузер технологию Web Storage. Эта подпрограмма является частью файла `browser-test.html`, находящегося в папке `code/storage`. Вместо этого кода можно воспользоваться библиотекой Modernizr, способной справляться с некоторыми случаями, приводящими к ложноположительному определению, такому, например, как ложное определение отсутствия поддержки хранилища в режиме инкогнито в Chrome.

Листинг 9.1. Проверка поддержки технологии Web Storage

```
function checkStorageSupport() {

  // хранилище сеанса
  if (window.sessionStorage) {
    alert('Этот браузер поддерживает sessionStorage');
  } else {
    alert('Этот браузер НЕ поддерживает sessionStorage');
  }

  // локальное хранилище
  if (window.localStorage) {
    alert('Этот браузер поддерживает localStorage');
  } else {
    alert('Этот браузер НЕ поддерживает localStorage');
  }
}
```

Результат проверки показан на рис. 9.1.

Многие браузеры не поддерживают сеансовое хранилище для файлов, доступ к которым осуществляется непосредственно из файловой системы. При выполнении примеров, приведенных в этой главе, убедитесь в том, что страница обслуживается сервером. Например, можете запустить простой HTTP-сервер среды разработки Python в папке `code/storage` с помощью следующей команды:

```
python -m SimpleHTTPServer 9999
```

После этого вы сможете получить доступ к файлам по адресу `http://localhost:9999/`. Например:

```
http://localhost:9999/browser-test.html
```

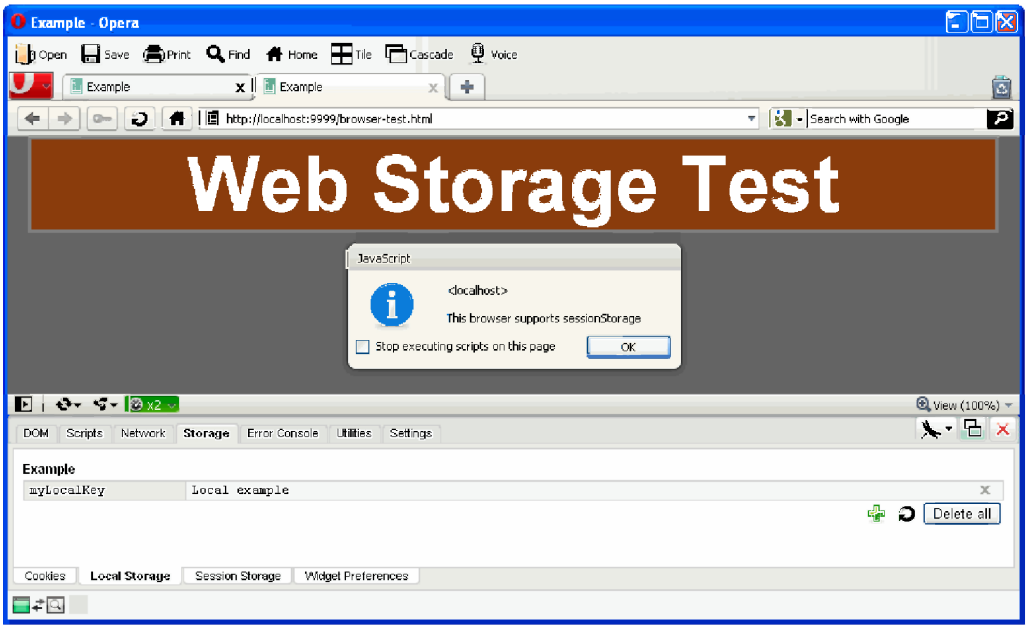


Рис. 9.1. Проверка поддержки в браузере

Примечание

Как и для многих других спецификаций, может случиться так, что браузер поддерживает стандарт лишь частично. Однако в силу того, что размеры библиотеки Web Storage API не очень велики, совместимые реализации получили весьма широкое распространение. Также необходимо учитывать, что даже при наличии поддержки Web Storage некоторые пользователи могут отключить использование этой библиотеки в своих браузерах из соображений безопасности.

Сохранение и извлечение значений

Перейдем к рассмотрению возможности сохранения сеансовых данных и посмотрим, как сохраняются и извлекаются простые значения на странице. Сохранение значения легко выполнить с помощью единственного оператора. Ниже приведен соответствующий пример, в котором используется длинная нотация.

```
window.sessionStorage.setItem('myFirstKey', 'myFirstValue');
```

Относительно этой строки можно сделать три важных замечания.

- Объект, реализующий интерфейс Web Storage, присоединяется к окну, и поэтому объект `window.sessionStorage` содержит все функции, которые вам требуется вызывать.
- Вызываемой функцией является функция `setItem()`, принимающая строку ключа и строку значения в качестве аргументов . Несмотря на то что формат данных Web Storage поддерживает передачу значений, не являющихся строковыми, в текущих версиях браузеров набор поддерживаемых типов значений ограничен.

- Данный конкретный вызов помещает в хранилище сеанса строку `myFirstValue`, которую впоследствии можно будет извлечь по ключу `myFirstKey`.

Извлечение значений с использованием длинной нотации осуществляется с помощью функции `getItem()`. Так, если мы дополним предыдущий пример следующим оператором:

```
alert(window.sessionStorage.getItem('myFirstKey'));
```

то браузер выведет диагностическое окно JavaScript, в котором отобразится текст `myFirstValue`. Как видите, занесение данных в хранилище и их извлечение из хранилища средствами Web Storage не составляет большого труда.

Тем не менее существует еще более простой способ доступа к хранилищам из программного кода. Для сохранения значений в хранилище можно использовать так называемые *expando-свойства* (расширяемые свойства). При таком подходе можно полностью избежать вызова функций `setItem()` и `getItem()`, сохраняя и извлекая значения, соответствующие парам *ключ-значение*, непосредственно в объекте `sessionStorage`. Например, для сохранения значения можно использовать следующий вызов:

```
window.sessionStorage.myFirstKey = 'myFirstValue';
```

Точно так же, для извлечения значения вызов можно переписать следующим образом:

```
alert(window.sessionStorage.myFirstKey);
```

Для начала этого достаточно. Теперь вы знаете все, что может понадобиться для использования хранилища сеанса. Однако может возникнуть вопрос: а что такого особенного предлагает объект `sessionStorage`? В конце концов, JavaScript позволяет устанавливать и получать значения свойств практически для любого объекта. Различие заключается в области действия. Возможно, вы не обратили внимания на тот факт, что в нашем примере сохранения и извлечения значений соответствующие вызовы функций не должны обязательно относиться к одной и той же вебстранице. Коль скоро обслуживаемые страницы относятся к одному и тому же источнику, т.е. адресуются с использованием одного и того же сочетания схемы, хоста и порта, то значения, сохраненные в объекте `sessionStorage`, могут извлекаться из других страниц с помощью тех же ключей. То же самое относится и к ряду последовательных загрузок одной и той же страницы. Как разработчик, вы, вероятно, уже свыклись с мыслью о том, что изменения, внесенные из сценария, исчезают при всякой перезагрузке страницы. К значениям, установленным с помощью технологии Web Storage, это не относится, и они продолжают существовать от одной загрузки страницы к другой.

Нарушения области действия данных

Как долго существуют сохраненные значения? В случае сеансового хранилища они будут существовать, пока не закроется окно (или вкладка). Как только пользователь закроет окно (или браузер), значения, сохраненные в объекте `sessionStorage`, пропадают. Такой способ сохранения значений чем-то напоминает стикеры для памятных записок. Значения, помещенные в сеансовое хранилище, долго не существуют, и поэтому в нем не следует сохранять критически важную информацию, ведь в нужный момент ее может не оказаться под рукой.

В чем же тогда состоит смысл использования сеансовых хранилищ в вебприложениях? Дело в том, что такое хранилище прекрасно подходит для "короткоживущих" процессов, с которыми обычно приходится сталкиваться в программах-мастерах и диалоговых окнах. Если данные должны храниться лишь в те-

чение времени, необходимого для просмотра нескольких страниц, к которым пользователь вряд ли вернется при следующем посещении вашего приложения, то сеансовое хранилище вполне подойдет для их сохранения. В прошлом такие данные отправлялись вместе с формой или в виде файла "cookie" и пересылались между клиентом и сервером при каждой загрузке страницы. Использование хранилища позволяет избежать ненужного трафика.

У объекта `sessionStorage` есть еще одно, весьма специфическое применение, позволяющее решить проблему, от которой страдали многие веб-приложения: область действия значений. В качестве примера рассмотрим приложение, позволяющее заказывать авиабилеты. В этом приложении данные, описывающие предпочтения пользователя, например наиболее желательные даты вылета и возвращения, могут пересылаться между браузером и сервером с помощью файлов "cookie". Это позволяет серверу запоминать ранее введенные пользователем данные по мере того, как он перемещается по приложению, выбирая, например, конкретное место в салоне самолета.

Однако очень часто пользователи, сравнивая условия полета, предлагаемые различными авиакомпаниями на определенную дату, открывают сразу несколько окон. Из-за этого в системе обработки файлов "cookie" могут возникать проблемы, ибо если пользователь в процессе выяснения наличия билетов и цен на них у различных перевозчиков многократно переходит из одного окна браузера в другое, то возрастает вероятность того, что файлы "cookie", установленные в одном окне, неожиданно будут применены к другому окну, обслуживаемому с того же URL-адреса, при следующем обращении к нему. Такое поведение, иногда называемое *нарушением области действия данных* (leaking data), может наблюдаться из-за того, что файл "cookie" совместно используется всеми страницами, связанными с источником, в котором он хранится.

Рис. 9.2 иллюстрирует, как все это может происходить.

Сравнение локального хранилища и хранилища сеанса

Иногда приложению могут потребоваться значения, которые продолжают существовать после того, как закроется отдельная вкладка или окно, или значения, которые должны совместно использоваться несколькими различными представлениями. В подобных случаях вам больше подойдет другая реализация интерфейса HTML5 Web Storage: `localStorage`. С процедурой использования локального хранилища вы уже фактически познакомились. С точки зрения программирования различие в использовании сеансового и локального типов хранилищ сводится к различию имен объектов, посредством которых осуществляется доступ к ним: `sessionStorage` и `localStorage` соответственно. С точки же зрения их поведения они отличаются главным образом сроком жизни и характером совместного использования сохраняемых в них значений. Различия между двумя типами хранилищ указаны в табл. 9.2.

Таблица 9.2. Различия между хранилищами `sessionStorage` и `localStorage`

`sessionStorage`	`localStorage`
Значения существуют до тех пор, пока открыты окно или вкладка, в которых они сохранены	Значения существуют независимо от закрытия окна или браузера
Значения видимы лишь в окне или вкладке, в котором они были созданы	Значения совместно используются всеми окнами и вкладками, выполняющимися на том же источнике

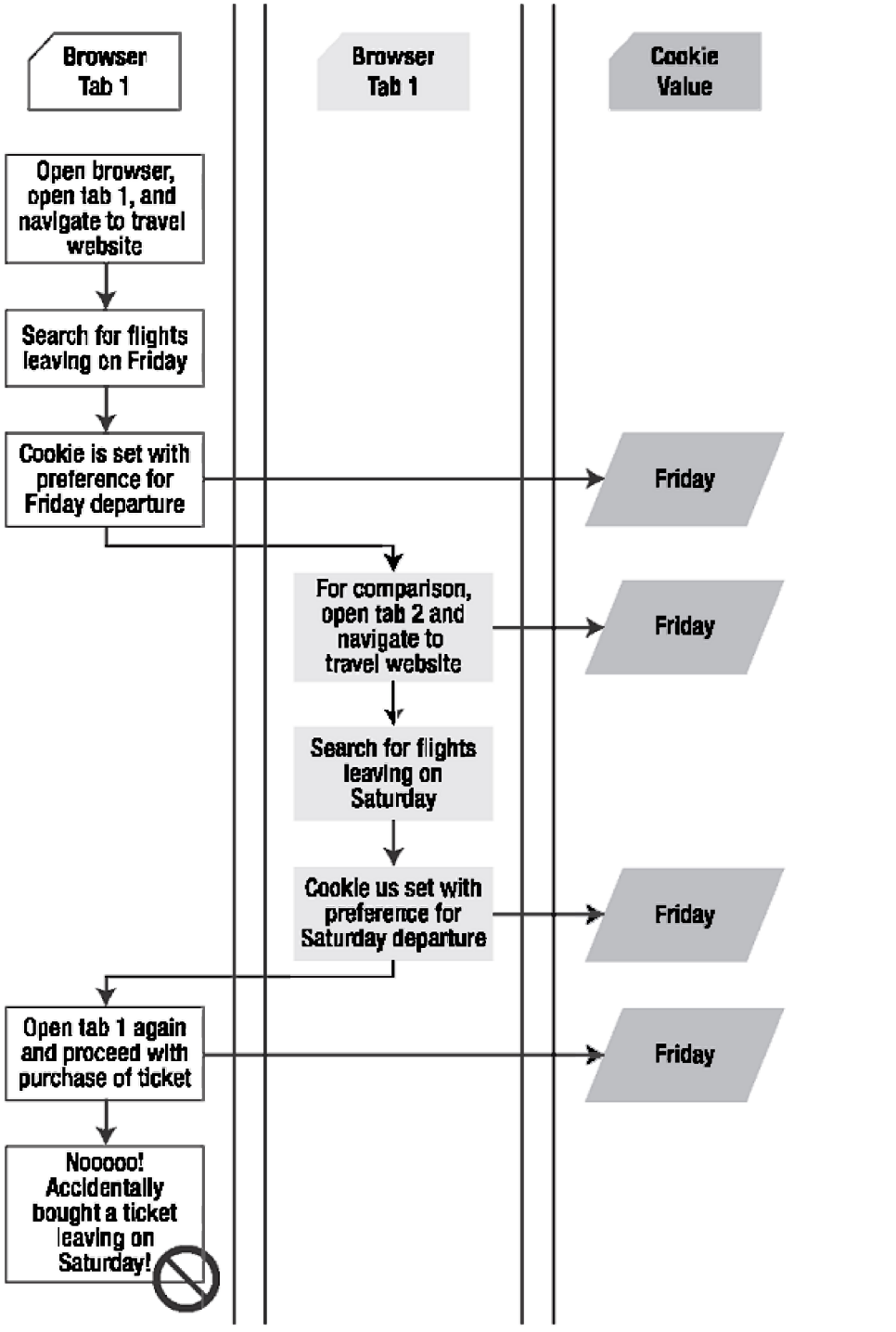


Рис. 9.2. Нарушение области действия данных в процессе сравнения цен на различных страницах сайта туристического бюро

Другие атрибуты и функции Web Storage

Программный интерфейс HTML5 Web Storage — один из простейших в HTML5. Нами уже рассмотрены как явный, так и неявный способы сохранения и извлечения данных в случае сеансового и локального хранилищ. Завершим наш обзор обсуждением полного набора доступных атрибутов и функций.

Объекты `sessionStorage` и `localStorage` можно получать с помощью объекта `window` документа, в котором они используются. Они обладают одинаковой функциональностью, если не считать различий в их именах и сроках существования. Оба они реализуют интерфейс `Storage`, представленный в листинге 9.2.

Листинг 9.2. Интерфейс `Storage`

```
interface Storage {
    readonly attribute unsigned long length;
    getter DOMString key(in unsigned long index);
    getter any getItem(in DOMString key);
    setter creator void setItem(in DOMString key, in any data);
    deleter void removeItem(in DOMString key);
    void clear();
};
```

Перейдем к более подробному рассмотрению атрибутов и функций.

- Атрибут `length` указывает количество пар *ключ-значение*, которые в данный момент хранятся в объекте хранилища. Не забывайте о том, что для каждого источника имеется свой объект хранилища, т.е. элементы (вместе с их размерными характеристиками), содержащиеся в объекте хранения, представляют лишь элементы, сохраненные для текущего источника.
- Функция `key(index)` обеспечивает извлечение заданного ключа. Необходимость в этом чаще всего возникает при итерации по всем ключам в конкретном объекте хранилища. Ключи индексируются, начиная с нуля, т.е. первому ключу соответствует индекс `(0)`, а последнему — индекс `(length-1)`. Как только получен ключ, его можно использовать для получения соответствующего значения. Ключи сохраняют свои индексы на протяжении всего срока существования данного объекта хранилища, если только не удаляется сам ключ или один из его предшественников.
- Как вы уже видели, одним из способов извлечения значений, индекс которых известен, является вызов функции `setItem(key, value)`. Другой способ состоит в использовании ключа в качестве индекса массива в объекте хранилища. В обоих случаях, если указанного ключа в хранилище не существует, то возвращается значение `null`.
- Аналогичным образом функция `setItem(key, value)` помещает значение с указанным именем ключа в хранилище или замещает уже существующее значение с тем же именем ключа. Имейте в виду, что при сохранении значений в хранилище также возможно возникновение ошибок. В случае отключения хранилища для данного сайта пользователем, а также при исчерпании памяти хранилища любая попытка сохранения в нем значений вызывает ошибку `QUOTA_EXCEEDED_ERR` (превышение квоты). Если работа вашего приложения зависит от нормального функционирования хранилища, обязательно предусмотрите обработку этой ошибки.
- Функция `removeItem(key)` делает именно то, о чем говорит ее название. Если в текущий момент времени значение с определенным ключом находится

в хранилище, то вызов этой функции удалит его. Если в хранилище отсутствует значение с данным ключом, то никаких действий не предпринимается.

Примечание

В отличие от некоторых коллекций и структур данных, удаление элемента не сопровождается возвращением старого значения в качестве результата вызова функции `removeItem()`. Поэтому вы должны самостоятельно позаботиться о сохранении копии удаленного значения, не полагаясь на вызов данной функции.

- Наконец, функция `clear()` удаляет из хранилища весь список значений. Вызывая эту функцию для пустого объекта хранилища, вы ничем не рискуете, поскольку при этом просто не будут выполнены никакие действия.

Квоты дискового пространства

Говорит Питер: "В спецификации рекомендуется, чтобы браузеры разрешали использовать под хранилище 5 Мбайт на один источник. Если места не хватает, браузеры должны предложить пользователю увеличить объем хранилища и, возможно, предоставить ему возможность увидеть, какой объем хранилища используется каждым из источников.

В действительности браузеры ведут себя в этом отношении по-разному. Некоторые браузеры разрешают увеличить размер хранилища или сами предлагают это сделать, другие просто генерируют ошибку `QUOTA_EXCEEDED_ERROR` (рис. 9.3). Тестовый файл `testQuota.html`, используемый в этом примере, находится в папке `code/storage`."

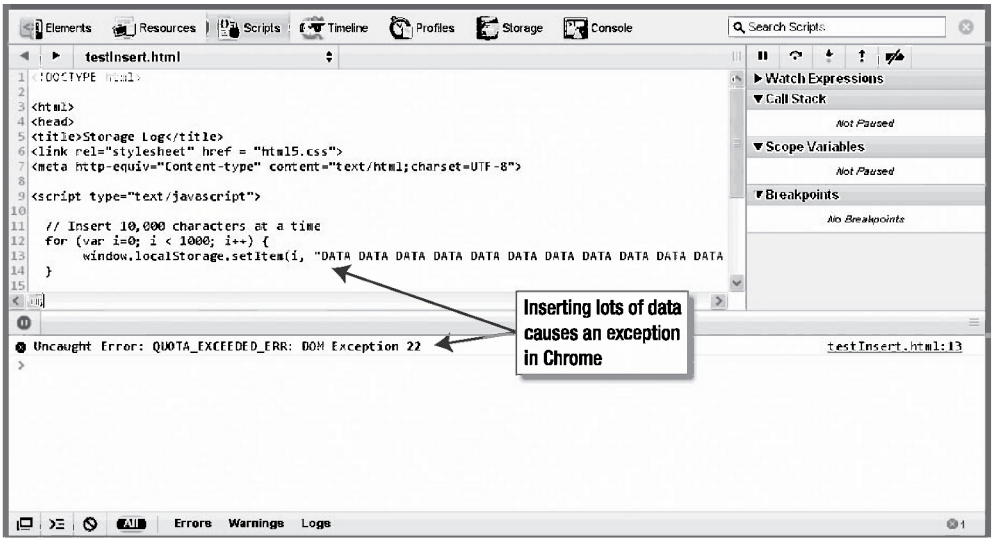


Рис. 9.3. Возникновение ошибки `QUOTA_EXCEEDED_ERROR` в браузере Chrome

Передача уведомлений об обновлениях Web Storage

Если доступ к хранилищу требуется нескольким страницам, вкладкам браузера или потокам выполнения, то ситуация несколько усложняется. Возможно, вашему приложению потребуется последовательно запускать множество операций при любом изменении одного из сохраненных значений. Именно для таких случаев в интерфейсе HTML5 Web Storage предусмотрен механизм событий, позволяющий вы-

сылать уведомления об обновлении данных нужным адресатам. События Web Storage запускаются в объектах window для каждого окна, источник которого совпадает с источником операции, выполняемой с хранилищем, независимо от того, выполняет или не выполняет какие-либо операции с хранилищем само окно, получающее события.

Примечание

События Web Storage могут использоваться для обмена сообщениями между окнами с одним и тем же источником. Более подробно об этом будет говориться в конце главы.

Для получения событий, относящихся к тому же источнику, что и окно, достаточно зарегистрировать обработчик событий, например:

```
window.addEventListener("storage", displayStorageEvent, true);
```

Здесь имя "storage" указывает на то, что мы заинтересованы в обработке событий хранилища. Каждый раз, когда для данного источника генерируется событие Storage (будь то sessionStorage или localStorage), его будет получать любой зарегистрированный обработчик. В листинге 9.3 представлен интерфейс объекта StorageEvent.

Листинг 9.3. Интерфейс объекта `StorageEvent`

```
interface StorageEvent : Event {
    readonly attribute DOMString key;
    readonly attribute any oldValue;
    readonly attribute any newValue;
    readonly attribute DOMString url;
    readonly attribute Storage storageArea;
};
```

Объект StorageEvent будет передан первым в обработчик событий, и в нем будет содержаться вся информация, которая необходима для определения природы произошедшего в хранилище изменения.

- Атрибут key содержит значение ключа, который был обновлен или удален из хранилища.
- Атрибут oldValue содержит прежнее значение, соответствующее данному ключу до того, как он был обновлен, тогда как атрибут newValue содержит измененное значение. Если значение было вновь добавлено, oldValue будет равно null, а если значение было удалено, то равным null будет newValue.
- Атрибут url указывает источник, в котором произошло событие.
- Наконец, атрибут storageArea предоставляет удобную ссылку на хранилище sessionStorage или localStorage, в котором было изменено значение. Используя эту ссылку, обработчик может легко запрашивать текущие значения из хранилища или вносить изменения на основании других изменений, произошедших в хранилище.

В листинге 9.4 представлен простой обработчик событий, который открывает диалоговое окно, отображающее содержимое любого события, сгенерированного на странице.

Листинг 9.4. Обработчик событий, отображающий содержимое объекта `StorageEvent`

```
// Вывести содержимое события хранилища
function displayStorageEvent(e) {
```

```
  var logged = "key:" + e.key + ", newValue:" + e.newValue +
               ", oldValue:" + e.oldValue +", url:" + e.url +
               ", storageArea:" + e.storageArea;
  alert(logged);
}

// Добавить обработчик событий хранилища для этого источника
window.addEventListener("storage", displayStorageEvent, true);
```

Просмотр хранилищ Web Storage в браузерах

Поскольку хранилища HTML5 Web Storage по своим функциям весьма напоминают файлы "cookie", то вас не должно удивлять, что в большинстве современных браузеров работа с ними организована примерно одинаковым образом. В самых последних версиях браузеров значения, содержащиеся в хранилищах `sessionStorage` и `localStorage`, можно просматривать наряду с файлами "cookie", как показано на рис. 9.4.

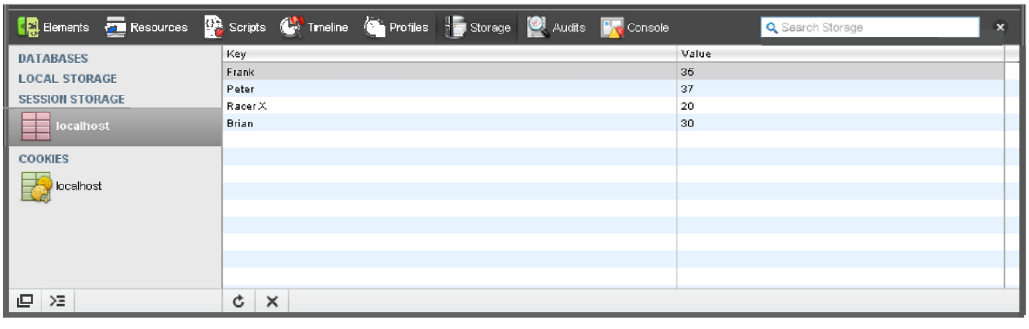


*Рис. 9.4. Отображение содержимого хранилища в панели **Storage** браузера Google Chrome*

Кроме того, этот интерфейс позволяет удалять значения из хранилища, а также наблюдать за тем, какие значения записываются в него данным сайтом при посещении страниц. Нет ничего удивительного в том, что браузер Safari компании Apple предлагает для просмотра файлов "cookie" и хранилища похожий унифицированный визуальный интерфейс, поскольку он основан на том же базовом движке визуализации WebKit, что и Chrome. Вид панели Storage в браузере Safari показан на рис. 9.5.

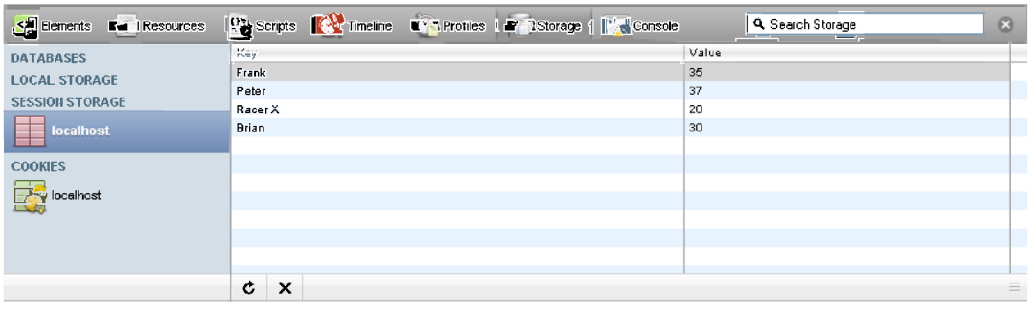


*Рис. 9.5. Отображение содержимого хранилища в панели **Storage** браузера Safari*

Браузеры Opera предоставляют дополнительные возможности, позволяя не только просматривать и удалять значения, но и сохранять значения в хранилище, как показано на рис. 9.6.

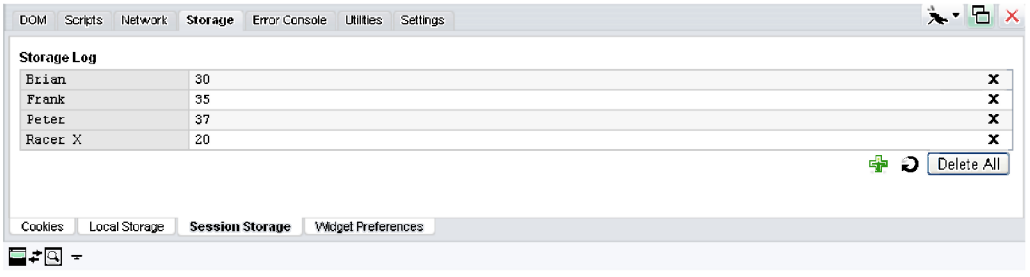


*Рис. 9.6. Отображение содержимого хранилища в панели **Storage** браузера Opera*

По мере того как технология Web Storage все шире внедряется различными производителями браузеров, можно ожидать быстрого увеличения емкости хранилищ и усовершенствования вспомогательных средств, доступных пользователям и разработчикам.

Создание приложения, использующего технологию HTML5 Web Storage

А теперь давайте закрепим те знания, которые вы приобрели к настоящему моменту, встроив хранилище в веб-приложение. По мере увеличения сложности приложений обработка как можно большего количества данных без взаимодействия с сервером становится все более актуальной. Локальное хранение данных на стороне клиента обеспечивает снижение сетевого трафика и улучшение интерактивных характеристик приложения, поскольку в этом случае данные не поступают от удаленного источника по сети, а извлекаются из локального хранилища на компьютере.

К числу общих проблем, с которыми приходится сталкиваться разработчикам, относится вопрос о том, как лучше всего управлять данными в приложении при переходе пользователя между страницами. Традиционным способом решения этой задачи в веб-приложениях является сохранение данных на сервере с последующей их пересылкой между сервером и клиентом в процессе навигации пользователя по страницам. Другой возможный способ предполагает исключение необходимости в переходах на другие страницы за счет динамического обновления всей необходимой информации на текущей странице. Однако пользователям свойственно переходить от одной веб-страницы к другой в процессе работы, и поэтому для улучшения интерактивности пользовательского интерфейса чрезвычайно важно, чтобы при возврате пользователя на страницу вашего приложения соответствующие данные могли быть отображены как можно быстрее.

В нашем приложении будет показано, как организовать локальное хранение временных данных при переходах пользователя от одной страницы веб-сайта к другой и обеспечить их быструю загрузку из хранилища на любую страницу. При создании приложения мы используем в качестве основы примеры, рассмотренные в предыдущих главах. Пример приложения, которое мы создали в главе 4, предназначался для демонстрации того, насколько просто осуществить сбор данных о текущем местоположении пользователя. В главе 6 мы расширили возможности этого

приложения, предусмотрев в нем отправку получаемых данных на удаленный сервер, чтобы сделать их доступными для просмотра любым количеством заинтересованных пользователей. Сейчас мы продвинемся еще дальше: доставляемые сервером через веб-сокет широковещательные данные о местоположении участников соревнований будут сохраняться в локальном хранилище, становясь доступными для немедленного просмотра при возврате пользователя на страницу приложения.

Итак, представьте, что информация о местоположении каждого участника соревнований, организованных неким вымышленным легкоатлетическим клубом, передается мобильными устройствами участников на сервер WebSocket, благодаря чему она становится доступной для коллективного использования. По мере поступления обновляемых данных от участников забега веб-приложение отображает текущее местоположение каждого участника в режиме реального времени. Хорошо продуманный веб-сайт должен позаботиться о кешировании этой информации, чтобы соответствующая страница могла быстро отображаться при возврате к ней пользователя в промежутках между посещениями других страниц сайта. Именно такое приложение мы и собираемся построить.

Для этого нам понадобится демонстрационный веб-сайт, который будет обеспечивать сохранение и извлечение данных об участниках забега. В данном примере, полный код которого вы найдете в папке `code/storage` на сайте книги, создается сайт с тремя страницами, однако для демонстрации рассматриваемых возможностей вы можете использовать любой другой сайт по своему усмотрению. Требуется лишь, чтобы в вашем распоряжении было несколько страниц, между которыми пользователь может совершать переходы. В эти страницы мы вставим динамическое содержимое в виде табло лидеров забега — списка участников вместе с данными о том, какой участок дистанции осталось пробежать до финишной черты каждому участнику. Три страницы, образующие сайт, представлены на рис. 9.7.

Рис. 9.7. Пример сайта забега

Каждая веб-страница будет содержать общий раздел, отображающий табло лидеров забега. В строках таблицы будут отображаться имена участников забега и расстояния, которые им остается пробежать до финишной черты. При загрузке любой из наших страниц она будет подключаться через веб-сокет к серверу, осуществляющему широковещательную рассылку данных о забеге, и получать сообщения с информацией о позициях участников. В свою очередь, каждый из участников забега будет посылать информацию о своем текущем местоположении на этот же широковещательный сервер, в результате чего данные будут поступать на страницу в реальном времени.

Все это уже рассматривалось нами в предыдущих главах, посвященных геолокации и веб-сокетам. Фактически, значительная часть приведенного в них демонстрационного кода будет включена в наш пример. Вместе с тем, создаваемое нами приложение имеет одно существенное отличие: при получении страницей данных

они будут сохраняться в локальном хранилище для последующего использования. Затем, всякий раз, когда пользователь будет переходить на новую страницу, данные будут извлекаться и отображаться еще до установления нового WebSocket-соединения. Благодаря этому данные могут передаваться между страницами без использования файлов "cookie" или установления соединения с сервером.

Как и в главе, посвященной веб-сокетам, наши сообщения с данными о местоположении участников передаются через Интернет в простом формате, упрощающем их чтение и синтаксический разбор. Для этого используется формат `String`, в котором отдельные порции данных — имя, широта, долгота — разделяются точкой с запятой (;). Например, участник с именем `Racer X`, находящийся в точке с широтой `37.20` и долготой `121.53`, будет идентифицироваться следующей строкой:

```
;Racer X;37.20;-121.53
```

Займемся теперь анализом самого кода. Каждая из наших страниц будет содержать один и тот же JavaScript-код для подключения к серверу через веб-сокет, обработки и отображения сообщений с данными о лидерах забега, а также сохранения и извлечения данных с помощью хранилища `sessionStorage`. А раз так, то в реальном приложении этот код был бы одним из первых кандидатов для включения в библиотеку JavaScript.

Прежде всего приведем несколько вспомогательных функций, с которыми вы ранее уже познакомились. Для расчета расстояний, которые каждому из участников остается пробежать до финишной черты, нам необходимы подпрограммы, позволяющие рассчитывать расстояние между двумя географическими точками, как показано в листинге 9.5.

Листинг 9.5. Функции расчета расстояний

```
// Функции для определения расстояния между двумя точками
// с известными значениями широты и долготы
function toRadians(num) {
  return num * Math.PI / 180;
}

function distance(latitude1, longitude1, latitude2, longitude2) {
  // R - радиус земного шара в километрах
  var R = 6371;

  var deltaLatitude = toRadians((latitude2-latitude1));
  var deltaLongitude = toRadians((longitude2-longitude1));
  latitude1 = toRadians(latitude1),
  latitude2 = toRadians(latitude2);

  var a = Math.sin(deltaLatitude/2) *
          Math.sin(deltaLatitude/2) +
          Math.cos(latitude1) *
          Math.cos(latitude2) *
          Math.sin(deltaLongitude/2) *
          Math.sin(deltaLongitude/2);

  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
  var d = R * c;
  return d;
}

// Широта и долгота финишной черты забега у озера Тахо
var finishLat = 39.17222;
var finishLong = -120.13778;
```

В этом уже знакомом нам наборе функций, которые использовались в главе 6, расстояние между двумя точками рассчитывается с помощью функции `distance()`. Детали и точность вычислений для нас существенного значения не имеют, и применяемая методика вполне подходит для нашего примера.

В последних двух строках задается широта и долгота местоположения финишной черты. Как будет показано далее, эти координаты будут сравниваться с поступающими данными о позициях участников для вычисления расстояний до финишной черты и определения на этом основании позиции участника на табло лидеров забега.

Рассмотрим теперь небольшой фрагмент HTML-разметки, используемой для отображения страницы.

```
<h2>Live T216 Leaderboard</h2>
<p id="leaderboardStatus">Leaderboard: Connecting...</p>
<div id="leaderboard"></div>
```

В этих строках кода объявляются элементы с идентификаторами `leaderboardStatus` и `leaderboard`. Первый из этих элементов предназначен для вывода информации о состоянии подключения к веб-сокету, а второй — для вывода самого табло (leaderboard). Вспомогательная функция, представленная в листинге 9.6, обеспечивает отображение на табло позиций участников, информация о которых поступает через веб-сокет.

Листинг 9.6. Вспомогательная функция для отображения информации о позициях участников соревнований

```
// Отображает на странице имя и оставшееся расстояние
// для указанного участника
function displayRacerLocation(name, distance) {
   // Найти HTML-элемент с данным ID.
   // Если такой элемент не существует, создать его.
   var incomingRow = document.getElementById(name);
   if (!incomingRow) {
      incomingRow = document.createElement('div');
      incomingRow.setAttribute('id', name);
      incomingRow.userText = name;

      document.getElementById("leaderboard").appendChild(incomingRow);
   }

   incomingRow.innerHTML = incomingRow.userText + " is " +
                           Math.round(distance*10000)/10000 +
                           " km from the finish line";
}
```

Эта вспомогательная функция просто отображает информацию, принимая в качестве аргументов имя участника забега и соответствующее расстояние до финишной черты. На рис. 9.8 показано, как выглядит табло лидеров забега на странице `index.html`.

Имена участников используются в коде двояким образом: каждое из них не только помещается в сообщение с информацией о данном участнике, но также используется для ссылки на уникальный элемент `div`, в котором хранится эта информация. Если элемент `div` для данного участника уже существует, мы находим его с помощью стандартной процедуры `document.getElementByID()`. Если же такого элемента пока еще нет на странице, мы создаем его и помещаем в контейнер `leaderboard`. В любом случае элемент `div`, соответствующий данному участнику,

обновляется для вывода самого последнего значения расстояния до финишной черты, что приводит к немедленному обновлению данных, отображаемых на странице. Если вы читали главу 6, то все это должно быть вам уже знакомо из примера приложения, которое в ней создавалось.

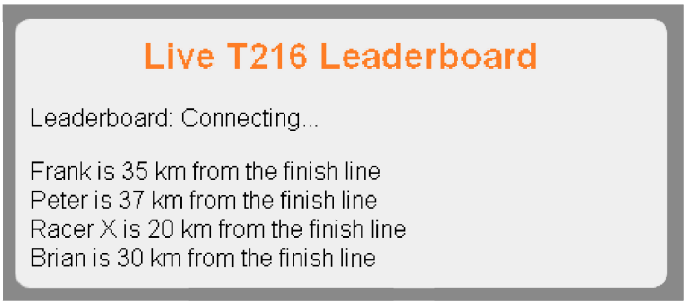


Рис. 9.8. Табло лидеров забега

Наша следующая функция — это обработчик сообщений, который вызывается всякий раз при получении данных широковещательной рассылки от сервера, как показано в листинге 9.7.

Листинг 9.7. Функция обработки сообщений, полученных через веб-сокет

```
// Автоматически вызывается при получении из веб-сокета
// новых данных о местоположении
function dataReturned(locationData) {
    // Разбить данные на ID, широту и долготу
    var allData = locationData.split(";");
    var incomingId   = allData[1];
    var incomingLat  = allData[2];
    var incomingLong = allData[3];

    // Обновить текст в строке новыми значениями
    var currentDistance = distance(incomingLat, incomingLong,
                                   finishLat, finishLong);

    // Сохранить полученные имя и расстояние в хранилище
    window.sessionStorage[incomingId] = currentDistance;

    // Отобразить на странице обновленные данные о пользователе
    displayRacerLocation(incomingId, currentDistance);
}
```

Эта функция принимает строку сообщения в ранее описанном формате, содержащую разделенные точкой с запятой имя участника, а также широту и долготу его местоположения. Прежде всего, мы должны разбить строку на отдельные компоненты для получения значений `incomingID`, `incomingLat` и `incomingLong`, используя JavaScript-функцию `split()`.

Затем эта функция передает широту и долготу местоположения участника, а также широту и долготу финишной черты определенному ранее вспомогательному методу `distance()`, сохраняя результирующее расстояние в переменной `currentDistance`.

Теперь, когда у нас действительно имеются данные, которые стоило бы сохранить, взглянем на вызов функции, обеспечивающей доступ к хранилищу HTML5 Web Storage.

```
// Сохранить в хранилище имя и оставшееся расстояние для участника
window.sessionStorage[incomingId] = currentDistance;
```

В этой строке объект `sessionStorage` окна используется для сохранения текущего расстояния от бегуна до финишной черты в виде значения, соответствующего имени и идентификатору бегуна. Иными словами, мы сохраняем значение в хранилище сеанса, используя в качестве ключа имя участника забега, а в качестве значения — его расстояние до финиша. Как вы сейчас увидите, эти данные будут извлекаться из хранилища при перемещении пользователя от одной страницы веб-сайта к другой. В конце этой функции вызывается ранее определенная процедура `displayLocation()`, обеспечивающая визуальное отображение на текущей странице обновленной информации о текущем местоположении участника соревнований.

Перейдем к рассмотрению последней функции, входящей в состав нашего приложения, — подпрограмме загрузки, представленной в листинге 9.8, которая запускается, когда посетители получают доступ к веб-странице.

Листинг 9.8. Подпрограмма начальной загрузки страницы

```
// Устанавливает соединение с широковещательным сервером
// при загрузке страницы
function loadDemo() {
    // Убедиться, что браузер поддерживает объект sessionStorage
    if (typeof(window.sessionStorage) === "undefined") {
        document.getElementById("leaderboardStatus").innerHTML =
                  "Ваш браузер не поддерживает HTML5 Web Storage";
        return;
    }
    var storage = window.sessionStorage;
    // Отобразить на странице новое местоположение бегуна для
    // каждого из хранящихся в базе данных ключей
    for (var i=0; i < storage.length; i++) {
        var currRacer = storage.key(i);
        displayRacerLocation(currRacer, storage[currRacer]);
    }

    // Убедиться в наличии поддержки веб-сокетов
    if (window.WebSocket) {

        // Расположение широковещательного WebSocket-сервера
        url = "ws://websockets.org:7999/broadcast";
        socket = new WebSocket(url);
        socket.onopen = function() {
           document.getElementById("leaderboardStatus").innerHTML =
                                          "Leaderboard: Connected!";
        }
        socket.onmessage = function(e) {
            dataReturned(e.data);
        }
    }
}
```

Эта функция содержит немного больше кода, чем остальные, и поэтому нам есть с чем разбираться. Сделаем это поэтапно. Сначала, как показано в листинге 9.9,

мы выполняем элементарную проверку, цель которой — удостовериться в том, что браузер, с помощью которого просматривается страница, поддерживает объект sessionStorage. Это легко сделать, проверив наличие объекта sessionStorage в объекте окна. Если объект sessionStorage недоступен, мы просто обновляем область leaderboardStatus, выводя в ней соответствующее сообщение, и возвращаемся из подпрограммы загрузки. Искать какие-либо обходные пути для обработки ситуаций, в которых хранилище браузером не поддерживается, в этом примере мы не будем.

Листинг 9.9. Проверка поддержки в браузере

```
// Убедиться, что браузер поддерживает объект sessionStorage
if (typeof(window.sessionStorage) === "undefined") {
    document.getElementById("leaderboardStatus").innerHTML =
                "Ваш браузер не поддерживает HTML5 Web Storage";
    return;
}
```

Примечание

Если хранилище не поддерживается браузером, то этот демонстрационный пример можно модифицировать таким образом, чтобы вообще отказаться от сохранения данных в промежутках между посещениями страниц и начинать загрузку каждой страницы с пустым табло лидеров. Однако наша основная цель состоит в том, чтобы продемонстрировать, каким образом использование хранилища помогает оптимизировать интерактивные свойства приложения как по отношению к пользователю, так и по отношению к сети.

Загрузив страницу, мы используем хранилище как для сохранения результатов, так и для извлечения тех результатов, которые были сохранены ранее при обслуживании этой или других страниц веб-сайта. Вспомните, что один и тот же блок кода сценария выполняется на каждой из страниц сайта, и поэтому табло лидеров сопровождает пользователей в процессе их путешествий по сайту. Следовательно, если обновленные данные табло лидеров уже были сохранены другими страницами в хранилище, то они могут быть извлечены и отображены сразу же после загрузки данной страницы, как показано в листинге 9.10.

Листинг 9.10. Отображение сохраненных данных об участнике соревнований

```
var storage = window.sessionStorage;

// Отобразить на странице новое местоположение бегуна для
// каждого из хранящихся в базе данных ключей
for (var i=0; i < storage.length; i++) {
    var currRacer = storage.key(i);
    displayRacerLocation(currRacer, storage[currRacer]);
}
```

Эта часть кода очень важна. Здесь мы запрашиваем размер сохраненных в сеансе данных, иными словами — количество ключей, содержащихся в хранилище. После этого мы получаем значение каждого ключа с помощью вызова storage.key() и сохраняем его в переменной currRacer, используя впоследствии эту переменную для обращения к значению, соответствующему данному ключу, с помощью выражения storage[currRacer]. Совместно используемые ключ и соответствующее ему значение представляют имя участника и его расстояние до финиша, сохраненные во время посещения предыдущей страницы.

Получив ранее сохраненные значения имени и расстояния, мы отображаем их с помощью функции displayRacerLocation(). Все это происходит очень быстро

при загрузке страницы, в результате чего табло лидеров, находящееся на данной странице, немедленно заполняется ранее переданными значениями.

Примечание

В данном примере предполагается, что хранилище сеанса используется только нашим приложением. Если бы хранилище одновременно использовалось для сохранения других данных, то для работы с ним потребовалась бы более избирательная стратегия в отношении ключей, чем их простое сохранение в корневом каталоге. С примером другой стратегии вы познакомитесь в конце главы.

Оставшаяся часть кода, выполняющегося при загрузке страницы, обеспечивает подключение страницы к широковещательному серверу с помощью простого веб-сокета, как показано в листинге 9.11.

Листинг 9.11. Подключение к широковещательной службе через веб-сокет

```
// Убедиться в наличии поддержки веб-сокетов
if (window.WebSocket) {

    // Расположение широковещательного WebSocket-сервера
    url = "ws://websockets.org:7999/broadcast";
    socket = new WebSocket(url);
    socket.onopen = function() {
        document.getElementById("leaderboardStatus").innerHTML =
            "Leaderboard: Connected!";
    }
    socket.onmessage = function(e) {
        dataReturned(e.data);
    }
}
```

Как мы уже делали в главе, посвященной веб-сокетам, сначала убеждаемся в том, что браузер поддерживает протокол WebSocket, проверив существование объекта `window.WebSocket`. Удостоверившись в том, что этот объект существует, подключаемся к URL-адресу, на котором выполняется наш сервер WebSocket. Этот сервер осуществляет широковещательную рассылку сообщений с информацией о местоположении участников в ранее описанном формате, роль разделителя в котором играет точка с запятой (;). Каждый раз при получении одного из этих сообщений посредством функции обратного вызова `socket.onmessage` мы вызываем для его обработки и отображения ранее обсуждавшуюся функцию `dataReturned()`. Мы также используем функцию обратного вызова `socket.onopen()` для обновления области `leaderboardStatus` простым диагностическим сообщением, указывающим на то, что сокет успешно открыт.

На этом рассмотрение подпрограммы `load()` завершено. В последнем блоке кода функция `loadDemo()` добавляется в число обработчиков событий загрузки страницы.

```
// Добавить обработчик событий загрузки страницы
window.addEventListener("load", loadDemo, true);
```

Аналогично тому, с чем вы уже многократно встречались в приведенных до этого примерах, зарегистрированная таким образом функция `loadDemo()` будет вызываться всякий раз при завершении процесса загрузки в окне.

Но каким образом данные с места проведения забега попадают сначала на веб-сервер, а затем на наши страницы? В принципе, мы могли бы воспользоваться примером приложения для отслеживания местоположения, которое рассматривалось нами в главе 6, указав в нем URL-адрес для подключения к широковещатель-

ному серверу, что также уже рассматривалось нами. В конце концов, все форматы данных у нас одинаковые. Однако того же результата можно добиться с помощью очень простой страницы, предоставляющей исходные данные для широковещательной рассылки, код которой приведен в листинге 9.12. Теоретически эта страница должна была бы выполняться на мобильных устройствах участников забега. Хотя она и не включает никакого кода, связанного собственно с HTML5 Web Storage, зато обеспечивает удобный способ передачи надлежащим образом отформатированных данных при выполнении в браузере, поддерживающем как протокол WebSocket, так и спецификацию Geolocation API. Файл `racerBroadcast.html` доступен на веб-сайте книги.

Листинг 9.12. Содержимое файла `racerBroadcast.html`

```
<!DOCTYPE html>

<html>

<head>
<title>Racer Broadcast</title>
<link rel="stylesheet" href="styles.css">
</head>

<body onload="loadDemo()">

<h1>Racer Broadcast</h1>

Racer name: <input type="text" id="racerName" value="Racer X"/>
<button onclick="startSendingLocation()">Start</button>

<div><strong>Geolocation</strong>: <p id="geoStatus">
HTML5 Geolocation not started.</p></div>
<div><strong>WebSocket</strong>: <p id="socketStatus">
HTML5 Web Sockets are <strong>not</strong> supported
in your browser.</p></div>

<script type="text/javascript">
    // Ссылка на веб-сокет
    var socket;

    var lastLocation;

    function updateSocketStatus(message) {
        document.getElementById("socketStatus").innerHTML = message;
    }

    function updateGeolocationStatus(message) {
        document.getElementById("geoStatus").innerHTML = message;
    }

    function handleLocationError(error) {
        switch(error.code)
        {
        case 0:
          updateGeolocationStatus("There was an error while
                                   retrieving your location: " +
                                   error.message);
          break;
```

```
    case 1:
      updateGeolocationStatus("The user prevented this page
                              from retrieving location.");
      break;
    case 2:
      updateGeolocationStatus("The browser was unable to
                              determine your location: " +
                              error.message);
      break;
    case 3:
      updateGeolocationStatus("The browser timed out before
                              retrieving the location.");
      break;
    }
}

function loadDemo() {
    // Убедиться в наличии поддержки веб-сокетов
    if (window.WebSocket) {
        // Расположение широковещательного WebSocket-сервера
        url = "ws://websockets.org:7999/broadcast";
        socket = new WebSocket(url);
        socket.onopen = function() {
            updateSocketStatus("Connected to WebSocket race
                               broadcast server");
        }
    }
}

function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var timestamp = position.timestamp;

    updateGeolocationStatus("Location updated at " + timestamp);

    // Подготовить сообщение для отправки данных
    // о местоположении через веб-сокет
    var toSend = ";" +
                 document.getElementById("racerName").value +
                 ";" + latitude + ";" + longitude;
    setTimeout("sendMyLocation('" + toSend + "')", 1000);
}

function sendMyLocation(newLocation) {
    if (socket) {
        socket.send(newLocation);
        updateSocketStatus("Sent: " + newLocation);
    }
}

function startSendingLocation() {
    var geolocation;
    if(navigator.geolocation) {
        geolocation = navigator.geolocation;
        updateGeolocationStatus("HTML5 Geolocation is supported
                                 in your browser.");
    }
    else {
```

```
            geolocation =
                  google.gears.factory.create('beta.geolocation');
            updateGeolocationStatus("Geolocation is supported via
                                    Google Gears");
        }

        // Зарегистрировать для обновления позиций
        // с использованием Geolocation API
        geolocation.watchPosition(updateLocation,
                                  handleLocationError,
                                  {maximumAge:20000});
    }
</script>
</body>
</html>
```

Мы не будем зря тратить время на подробный анализ кода этого файла, поскольку он практически совпадает с кодом приложения для отслеживания местоположения, рассмотренным в главе 6. Главное отличие состоит в том, что в данном файле содержится текстовое поле для ввода имени участника забега.

```
Racer name: <input type="text" id="racerName" value="Racer X"/>
```

Далее имя участника забега пересылается на широковещательный сервер в виде части строки данных.

```
var toSend = ";" + document.getElementById("racerName").value +
             ";" + latitude + ";" + longitude;
```

Чтобы проверить, как это работает на практике, откройте два окна в браузере, который поддерживает все три спецификации HTML5: Web Storage, Geolocation и WebSocket. В качестве такового можете использовать браузер Google Chrome. Во-первых, загрузите страницу сайта легкоатлетического клуба `index.html`. Вы увидите, как этот сайт подключится к широковещательному сайту через веб-сокет, а затем будет ожидать получения уведомлений с данными участников забега. Во втором окне откройте файл `racerBroadcast.html`. После того как и эта страница подключится к широковещательному сайту, введите имя участника и щелкните на кнопке Start (Пуск) Вы увидите, как сервер передаст данные о местоположении участника, которые должны отобразиться на табло лидеров в другом окне браузера. Как все это выглядит, показано на рис. 9.9.

Затем перейдите на одну из оставшихся страниц сайта клуба, воспользовавшись расположенными слева ссылками Signup (Зарегистрироваться) и About the Race (О соревнованиях) Поскольку все эти страницы сконфигурированы для загрузки нашего сценария, они немедленно загрузят и заполнят данными табло лидеров с предыдущими данными забега, которые были доставлены во время просмотра других страниц. Отправьте еще несколько сообщений о состоянии забега (со страницы широковещательного сервера), и вы увидите, что они распространятся на все страницы клубного сайта.

Теперь, когда мы рассмотрели весь код, целесообразно проанализировать в целом, что же мы все-таки создали. Мы создали простой функциональный блок, пригодный для включения в общую библиотеку JavaScript, который обеспечивает подключение к широковещательному серверу по протоколу WebSocket и получение обновляемой информации об участниках забега. При получении обновлений сценарий отображает позицию на странице *и* сохраняет ее в хранилище сеанса HTML5 `sessionStorage`. После загрузки страницы он сверяется с ранее сохраненными данными о местоположении участника забега, тем самым поддерживая со-

стояние при переходах пользователя в пределах сайта. Что же мы выиграли, применив описанный подход?

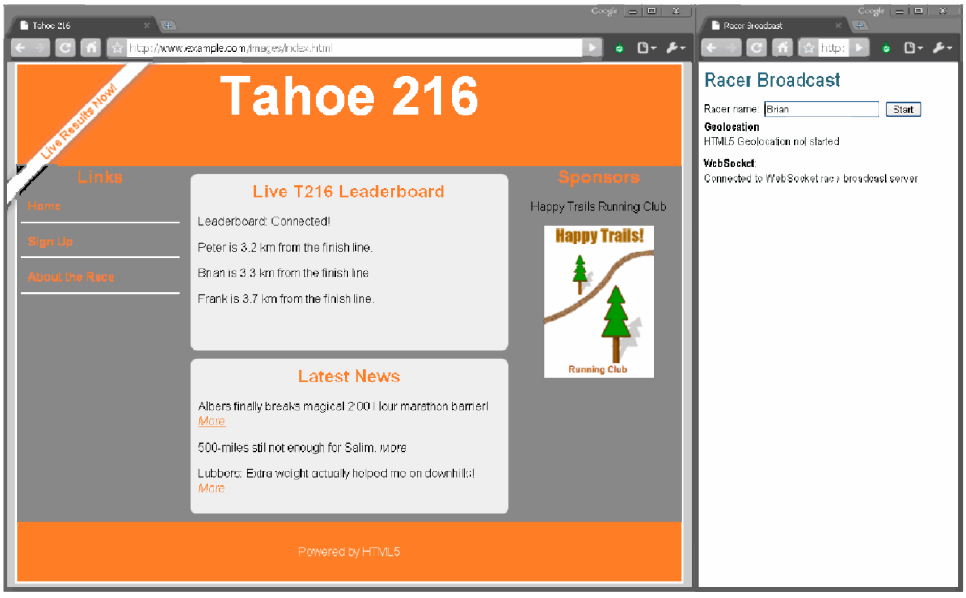


Рис. 9.9. Страница с информацией о забеге и страница `racerBroadcast.html`

- *Снижение сетевого трафика.* Информация о забеге сохраняется локально в браузере. Полученная информация все время находится под рукой даже без использования файлов "cookie" или обращений к серверу для ее повторного извлечения.
- *Немедленное отображение данных.* Сами страницы браузера могут кешироваться, а не загружаться по сети, поскольку динамически изменяемые части страницы — текущее состояние табло лидеров забега — представляются локальными данными. Эти данные быстро отображаются без каких-либо затрат времени на загрузку из сети.
- *Временное хранилище.* После того как соревнования закончились, данные о них не представляют особой ценности. Поэтому мы сохраняем их в хранилище сеанса, а это означает, что при закрытии окна или вкладки они будут уничтожены, тем самым освобождая занимаемые ими ресурсы.

Несколько слов о "пуленепробиваемости" программы

Говорит Брайан: "Нам удалось многого добиться в этом примере с помощью совсем небольшого по своему размеру сценарного кода. Но не думайте, что в случае реальных веб-сайтов, открытых для всеобщего доступа, все будет столь же просто. Мы выбирали кое-какие "короткие пути", которые для производственных приложений просто неприемлемы.

Например, формат наших сообщений не поддерживает участников с одинаковыми именами, и поэтому вместо имен было бы лучше использовать уникальные идентификаторы, представляющие каждого из участников. Наша методика расчета расстояний "по прямой" не совсем подходит для реального расчета расстояний в условиях пересеченной местности. В этом отношении остаются в силе стандартные оговорки — чтобы ваш сайт стал по-настоящему работоспособным, необходимо увеличить количество всевозможных проверок и уделить больше внимания деталям."

Та же самая методика, которая была продемонстрирована в этом примере, может использоваться по отношению к любым другим типам данных. Чат, электронная почта, подсчет очков в спортивных соревнованиях — все это примеры тех данных, которые можно кешировать и отображать на каждой странице, используя локальное или сеансовое хранилище, как это было сделано нами. Если ваше приложение регулярно пересылает специфические для пользователя данные между сервером и клиентом, то применение возможностей HTML5 Web Storage поможет вам упростить технологический процесс.

Будущее браузерных хранилищ

Пары *ключ-значение* в интерфейсе Web Storage великолепно подходят для сохранения данных, но что можно сказать об индексированных хранилищах, к которым можно обращаться с запросами? Для приложений HTML5 предусмотрен также доступ к индексируемым базам данных. Точные детали программного интерфейса баз данных пока еще только выкристаллизовываются, и к настоящему времени подано уже несколько предложений. Одно из них — Web SQL Database — реализовано в Safari, Chrome и Opera. Табл. 9.3 позволяет получить представление об уровне поддержки Web SQL Database в различных браузерах.

Таблица 9.3. Поддержка спецификации HTML5 Web SQL Database браузерами

Браузер	Описание
Chrome	Поддерживается в версии 3.0 и выше
Firefox	Не поддерживается
Internet Explorer	Не поддерживается
Opera	Поддерживается в версии 10.5 и выше
Safari	Поддерживается в версии 3.2 и выше

Используя Web SQL Database, приложения могут получать доступ к СУБД SQLite посредством асинхронного JavaScript-интерфейса. Хотя эта спецификация не является ни частью обычной веб-платформы, ни окончательно рекомендованным интерфейсом баз данных для HTML5, она может быть полезной, если ориентироваться на конкретные платформы, например мобильную версию Safari. Как бы то ни было, эта спецификация позволяет раскрыть всю мощь использования баз данных в браузерах. Как и в случае других интерфейсов, предназначенных для сохранения данных, браузер может ограничивать объем хранилища, доступный для каждого источника, и уничтожать данные, если уничтожаются пользовательские данные.

Судьба спецификации Web SQL Database

Говорит Фрэнк: "Несмотря на то что технология Web SQL Database уже поддерживается в Safari, Chrome и Opera, она не будет реализована в Firefox и числится на вики-странице группы WHATWG как спецификация, работа над которой остановлена ("stalled"). Спецификация определяет программный интерфейс для выполнения запросов, указываемых в виде строк, и ориентируется на диалект SQL для СУБД SQLite. Поскольку стандарт не должен определять конкретную реализацию SQL, Web SQL Database уступает место более новой спецификации, Indexed Database (прежнее название — WebSimpleDB), которая проще и не привязана к конкретной версии реляционной СУБД. В настоящее время реализации Indexed Database в браузерах находятся в стадии разработки."

Поскольку технология Web SQL Database уже частично реализована, мы включили элементарный пример ее использования, опустив полное рассмотрение соответствующего программного интерфейса. Этот пример демонстрирует базовое ис-

пользование Web SQL Database. В нем открывается база данных `mydb`, создается таблица `racer`, если таблицы с таким именем не существует, и эта таблица заполняется списком заданных имен. Вид базы данных с таблицей `racers` в окне средства Web Inspector Safari представлен на рис. 9.10.

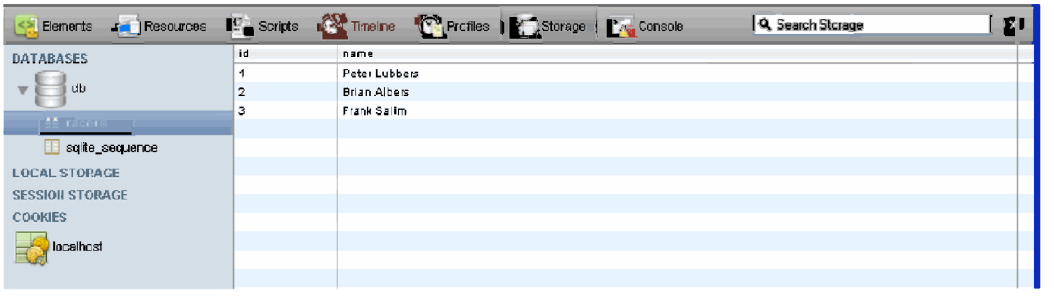


Рис. 9.10. База данных с таблицей `racers` в окне средства Web Inspector браузера Safari

Сначала мы открываем базу данных с указанным именем. Функция `window.openDatabase()` возвращает объект `Database`, через который осуществляется взаимодействие с базой данных. Функция `openDatabase()` принимает в качестве аргументов имя базы данных, а также необязательные номер версии и описание. После того как база данных открыта, код приложения может выполнять транзакции. SQL-запросы выполняются в контексте транзакции с использованием функции `transaction.executeSql()`. В нашем простом примере функция `executeSql()` используется для создания таблицы, вставки в нее имен участников забега и выполнения последующего запроса для создания HTML-таблицы. На рис. 9.11 представлен выходной HTML-файл со списком имен, извлеченных из таблицы.

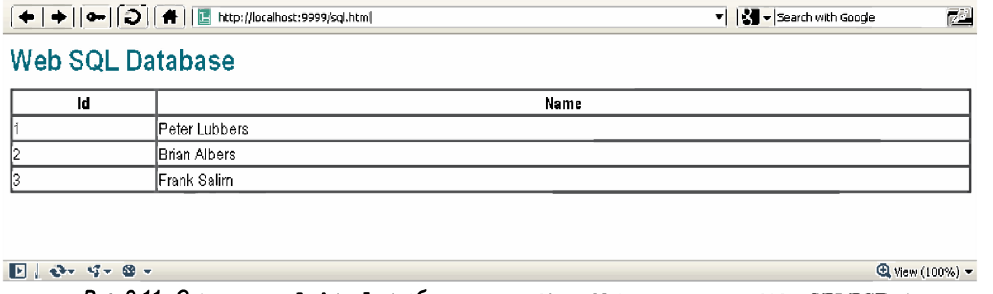


Рис. 9.11. Страница `sql.html`, отображающая результат выполнения запроса `SELECT * FROM racers`

Для завершения операций с базой данных требуется определенное время. Чтобы выполнение сценария не было заблокировано до тех пор, пока не станет доступным результирующий набор, запросы выполняются в фоновом режиме. Когда результаты становятся доступными, функция, указанная в качестве третьего аргумента при вызове функции `executeSql()`, вызывается как функция обратного вызова с транзакцией и результирующим набором в качестве аргументов.

Полный код этого примера, представленный в листинге 9.13, можно найти в папке `code/storage` (файл `sql.html`) на сайте книги.

Листинг 9.13. Использование технологии Web SQL Database

```
<!DOCTYPE html>
<title>Web SQL Database</title>
<script>

    // Открыть базу данных с указанным именем
    var db = openDatabase('db', '1.0', 'my first database',
                          2 * 1024 * 1024);

    function log(id, name) {
        var row = document.createElement("tr");
        var idCell = document.createElement("td");
        var nameCell = document.createElement("td");
        idCell.textContent = id;
        nameCell.textContent = name;
        row.appendChild(idCell);
        row.appendChild(nameCell);

        document.getElementById("racers").appendChild(row);
    }

    function doQuery() {
        db.transaction(function (tx) {
                tx.executeSql('SELECT * from racers', [],
                              function(tx, result) {
                    // Вывести результирующий SQL-набор
                    for (var i=0; i<result.rows.length; i++) {
                        var item = result.rows.item(i);
                        log(item.id, item.name);
                    }
                });
            });
    }

    function initDatabase() {
        var names = ["Peter Lubbers", "Brian Albers",
                     "Frank Salim"];

        db.transaction(function (tx) {
                tx.executeSql('CREATE TABLE IF NOT EXISTS racers
                              (id integer primary key
                               autoincrement, name)');

                for (var i=0; i<names.length; i++) {
                    tx.executeSql('INSERT INTO racers (name)
                                   VALUES (?)', [names[i]]);
                }

                doQuery();
            });
    }

    initDatabase();

</script>

<h1>Web SQL Database</h1>
```

```
<table id="racers" border="1" cellspacing="0" style="width:100%">
    <th>Id</th>
    <th>Name</th>
</table>
```

Дополнительные рекомендации

Некоторые методики, не нашедшие отражения в основных примерах, приведенных в данной книге, могут быть весьма полезными во многих типах приложений HTML5. Ниже представлены небольшие фрагменты кода, представляющие практический интерес, вместе с соответствующими комментариями и рекомендациями.

Сохранение объектов JSON

Несмотря на то что спецификация HTML5 Web Storage допускает сохранение объектов любого типа в виде пар *ключ-значение*, в текущих реализациях некоторые браузеры ограничивают типы значений строковыми данными. Тем не менее существует один практический прием, позволяющий обойти это ограничение. Он основан на том факте, что в современных браузерах предоставляется встроенная поддержка стандарта Java Script Object Notation (JSON).

JSON — это стандарт для обмена данными, позволяющий представлять объекты в виде строк и наоборот. Стандарт JSON используется для передачи объектов с браузеров клиентов на серверы по протоколу HTTP вот уже более десяти лет. Мы можем использовать его для сериализации сложных объектов при сохранении и извлечении данных с помощью технологии Web Storage для организации длительного хранения сложных типов данных. Рассмотрим блок кода сценария, представленный в листинге 9.14.

Листинг 9.14. Сохранение объекта JSON

```
<script>

  var data;

  function loadData() {
    data = JSON.parse(sessionStorage["myStorageKey"])
  }

  function saveData() {
    sessionStorage["myStorageKey"] = JSON.stringify(data);
  }

  window.addEventListener("load", loadData, true);
  window.addEventListener("unload", saveData, true);

</script>
```

Как видите, сценарий содержит код для регистрации обработчиков событий загрузки и выгрузки окна. В данном случае эти обработчики вызывают функции `loadData()` и `saveData()` соответственно.

В функции `loadData()` из хранилища сеанса запрашивается значение ключа, и этот ключ передается функции `JSON.parse()`. Функция `JSON.parse()` принимает ранее сохраненное строковое представление объекта и реконструирует из него копию оригинала. Эта функция вызывается при каждой загрузке страницы.

Точно так же функция saveData() принимает данные и вызывает для них функцию JSON.stringify(), преобразующую полученные данные в строковое представление объекта. Далее эта строка сохраняется в хранилище. Регистрируя функцию saveData() в качестве обработчика события браузера unload, мы гарантируем, что она будет вызываться всякий раз, когда пользователь покидает или закрывает окно или браузер.

Практический результат применения указанных двух функций состоит в том, что, каким бы сложным ни был тип объекта, который мы хотим отслеживать в хранилище, он может сохраняться и заново загружаться при перемещении пользователя между страницами как внутри приложения, так и за его пределами.

Совместное использование окон

Как уже отмечалось ранее, возможность запускать события HTML5 Web Storage в любом окне, из которого осуществляется просмотр того же источника, имеет очень важные последствия. Это означает, что хранилище может использоваться для отправки сообщений из одного окна в другое, даже если не все они используют хранимое значение. В свою очередь, это означает возможность совместного использования данных окнами, которые относятся к одному и тому же источнику.

Чтобы увидеть, как все это работает, обратимся к конкретному примеру. Для организации обмена сообщениями между окнами сценарий должен лишь зарегистрировать обработчик событий storage. Предположим, что страница, с адресом http://www.example.com/storageLog.html содержит код, представленный в листинге 9.15 (файл примера storage.html также находится в папке code/storage).

Листинг 9.15. Обмен сообщениями между окнами с помощью хранилища

```
// Отобразить записи новых событий хранилища
function displayStorageEvent(e) {
   var incomingRow = document.createElement('div');
   document.getElementById("container").appendChild(incomingRow);

   var logged = "key:" + e.key + ", newValue:" + e.newValue +
                ", oldValue:" + e.oldValue + ", url:" + e.url +
                ", storageArea:" + e.storageArea;
   incomingRow.innerHTML = logged;
}

// Добавить обработчик событий хранилища
window.addEventListener("storage", displayStorageEvent, true);
```

После регистрации обработчика событий хранилища данное окно будет получать уведомления об изменениях в хранилище, вызванных любой страницей. Например, если окно браузера, в котором осуществляется просмотр страницы http://www.example.com/browser-test.html, изменяет значение, установленное в хранилище, или записывает в хранилище новое значение, то страница storageLog.html получит уведомление (файл browser-test.html также находится в папке code/storage). Поэтому, чтобы отправить сообщение окну-получателю, окно-отправитель должно лишь изменить хранимое значение, и тогда его старое и новое значения будут отправлены в виде данных в составе уведомления. Например, если хранимое значение обновляется с помощью функции localStorage.setItem(), то обработчик displayStorageEvent() на странице storageLog.html, принадлежащей тому же источнику, получит событие. Тщательно согласовав имена и значения событий, можно добиться того, чтобы обе страницы могли обмениваться сообще-

ниями, что до сих пор сделать было очень непросто. На рис. 9.12 показана страница `storageLog.html`, которая в данном случае просто выводит информацию о получаемых ею событиях.

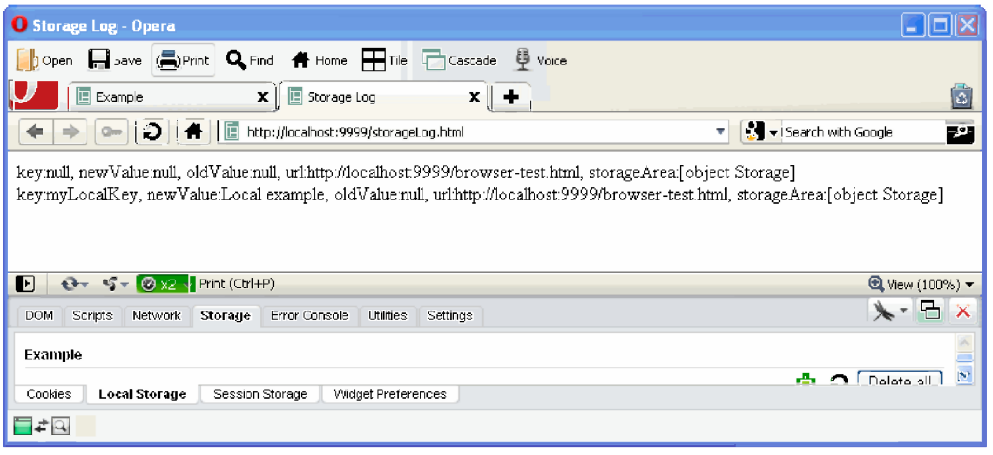


Рис. 9.12. Вывод страницей `storageLog.html` информации о получаемых ею событиях

Резюме

В этой главе было продемонстрировано, как использовать хранилище HTML5 Web Storage в качестве альтернативы файлам "cookie" для сохранения локальных копий данных при переходах между вкладками или окнами и даже (в случае объекта `localStorage`) при закрытии и повторном открытии браузера. Использование событий хранилищ позволяет изолировать данные в пределах окон с помощью объектов `sessionStorage` или обеспечить их совместное использование, причем даже разными окнами. В нашем примере полноценного приложения было продемонстрировано практическое использование хранилища для отслеживания данных при переходах пользователя между страницами в пределах сайта, что может применяться по отношению к данным любого типа. Было показано, что даже нетекстовые объекты данных могут взаимодействовать с хранилищем при загрузке или выгрузке страниц, тем самым обеспечивая сохранение и восстановление состояния в промежутках между посещениями.

В следующей главе демонстрируется создание автономных приложений средствами HTML5.

Глава 10

Создание автономных веб-приложений в HTML5

В этой главе рассматриваются возможности автономных приложений HTML5. Таким приложениям не требуется постоянный доступ к сети, а загрузка кешированных ресурсов теперь может контролироваться разработчиками гораздо более гибко.

Концепция автономных веб-приложений в HTML5

Первой и самой очевидной причиной использования кеша приложений является необходимость поддержки автономного режима. Потребность в автономных приложениях существует даже в наши дни, когда практически у каждого пользователя имеется возможность подключиться к сети. Но что делать в случае, если подключение к сети отсутствует? Прежде чем спешить заявлять, будто времена, когда постоянная связь с Интернетом была редкостью, давно канули в Лету, задумайтесь над следующим.

- Часто ли вам в авиаперелетах удавалось подключиться к бортовой сети Wi-Fi самолета?
- Идеально ли покрытие сети, к которой подключено ваше мобильное интернет-устройство (давно ли вы видели нулевую отметку сигнала)?
- Можете ли вы всегда рассчитывать на то, что во время презентации важного материала вам удастся беспрепятственно подключиться к Интернету?

Чем больше приложений перемещается в Интернет, тем соблазнительнее предполагать, что постоянное подключение к Интернету, работающее в режиме "24/7", доступно каждому пользователю. Однако реальность такова, что соединение может в любой момент прерваться, а в ситуациях, подобных перелетам, оно может отсутствовать в течение нескольких часов.

Ненадежность соединения является ахиллесовой пятой сетевых вычислительных систем. Если работа ваших приложений зависит от качества связи с удаленными хостами, а подключиться к ним не удается, то в ситуациях, подобных авиаперелетам, вам не повезет. В то же время, если у вас имеется подключение к Интернету, то веб-приложения всегда будут под рукой, поскольку каждый раз, когда используется их программный код, он загружается из удаленного источника.

Если же ваши приложения нуждаются лишь в периодической связи, то они попрежнему могут быть полезными, при условии, что будет организовано локальное хранение ресурсов. С появлением устройств, основным программным обеспечением которых является браузер, роль веб-приложений, способных нормально функционировать без постоянной связи с Интернетом, будет только возрастать. В этом отношении настольные приложения, для которых постоянное подключение к сети не требуется, исторически имели преимущество по сравнению с веб-приложениями.

HTML5 предлагает рычаги контроля над кешированием приложений, позволяя объединить преимущества двух сред: мы получаем построенные на основе веб-технологий приложения, которые выполняются в браузере и обновляются при подключении к сети, но при необходимости могут использоваться автономно. Однако к этим новым возможностям автономных приложений следует обращаться явным образом, поскольку текущие версии серверов не рассчитаны на предоставление кеширования по умолчанию, что необходимо для нормальной работы автономных приложений.

Кеширование автономных приложений HTML5 обеспечивает расширение их возможностей, позволяя выполнять их без подключения к сети. Чтобы написать черновик письма электронной почты, вам не нужно подключаться к Интернету. HTML5 вводит кеш автономных приложений, благодаря которому веб-приложения могут выполняться, не будучи подключенными к сети.

Разработчик приложения может конкретно указать дополнительные ресурсы, являющиеся компонентами приложения HTML5 (HTML, CSS, JavaScript и изображения) и делающие его доступным в автономном режиме. К числу возможных областей применения автономных веб-приложений относятся:

- чтение и подготовка сообщений электронной почты;
- подготовка документов;
- подготовка и отображение презентаций;
- создание списков неотложных задач.

Использование автономных хранилищ позволяет обойтись без обычных сетевых запросов, необходимых для загрузки приложений. Если манифест кеша обновлен, то браузер знает, что не нужно проверять, обновлены ли другие ресурсы, и большая часть приложения может очень быстро загрузиться из локального кеша. Кроме того, загрузка ресурсов из кеша (вместо генерации множества HTTP-запросов с целью проверки обновления ресурсов) экономит часть полосы пропускания, что особенно важно в случае мобильных веб-приложений. В настоящее время медленная загрузка является одним из факторов, которые делают веб-приложения менее выигрышными по сравнению с настольными. Кеширование позволяет сгладить эти отличия.

Кеш приложения предоставляет разработчику возможность явного управления кешированием. С помощью файла *манифеста кеша* можно группировать родственные ресурсы в логически связанные приложения. Эта необычайно мощная концепция способна наделять веб-приложения характеристиками настольных приложений. Вы сможете найти для нее массу применений в соответствии с собственными творческими замыслами.

Ресурсы, идентифицированные в файле манифеста кеша, образуют то, что называется *кешем приложения*, который используется браузером для постоянного хранения ресурсов (как правило, на жестком диске). Некоторые браузеры предоставляют пользователю возможность просмотра содержимого кеша. Например, страница `about:cache` в последней версии браузера Firefox отображает подробные сведения о кеше приложения и позволяет просматривать отдельные файлы, находящиеся в кеше, как показано на рис. 10.1.

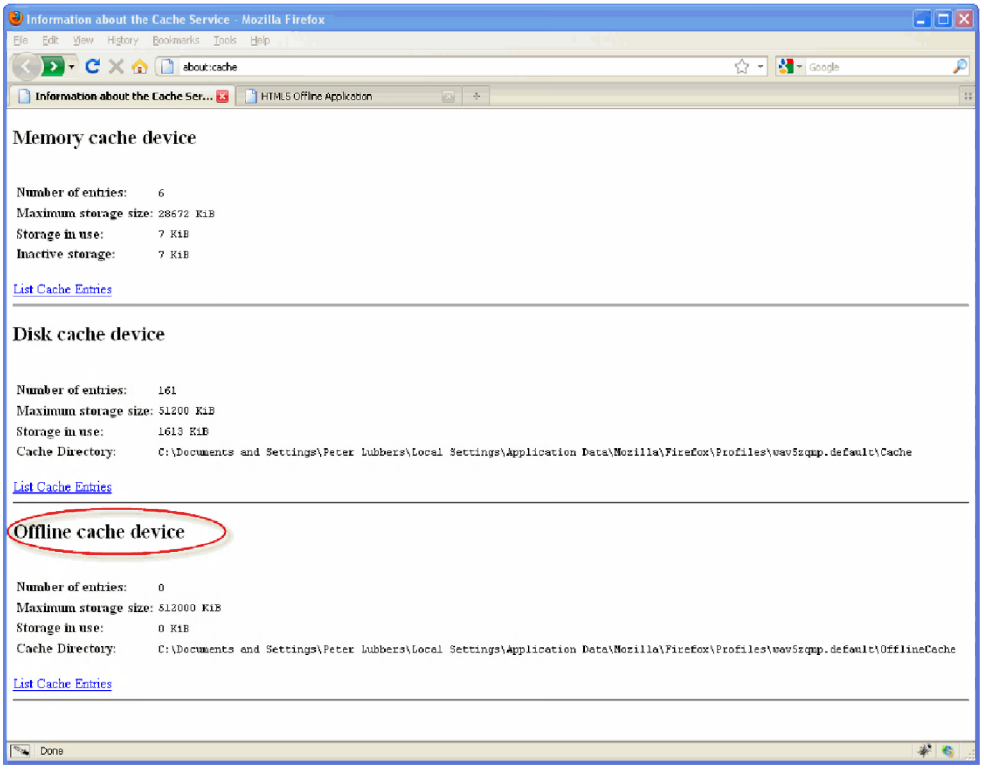


Рис. 10.1. Просмотр автономного кеша в Firefox

Поддержка автономных веб-приложений браузерами

Табл. 10.1 позволяет получить представление об уровне поддержки автономных веб-приложений HTML5, предоставляемой браузерами на момент написания книги. Как следует из приведенных здесь данных, приложения этого типа поддерживаются многими браузерами.

Таблица 10.1. Поддержка автономного кеша приложений HTML5 браузерами

Браузер	Описание
Chrome	Поддерживается в версии 4.0 и выше
Firefox	Поддерживается в версии 3.5 и выше
Internet Explorer	Не поддерживается
Opera	Поддерживается в версии 10.6 и выше
Safari	Поддерживается в версии 4.0 и выше

Поскольку уровень необходимой поддержки может быть разным в различных браузерах, то прежде чем использовать автономные веб-приложения HTML5, следует предварительно убедиться в том, что браузер их поддерживает.

Программный интерфейс автономных веб-приложений в HTML5

В этом разделе рассматривается специфика использования программного интерфейса автономных веб-приложений HTML5.

Проверка поддержки в браузере

Прежде чем пытаться использовать программный интерфейс автономных веб-приложений, не помешает проверить, предоставляет ли браузер необходимую поддержку. В листинге 10.1 показано, как это можно сделать.

Листинг 10.1. Проверка поддержки автономных веб-приложений в браузере

```
if(window.applicationCache) {
  // Автономные приложения поддерживаются данным браузером
}
```

Создание простого автономного приложения

Предположим, вы хотите создать одностраничное приложение, состоящее из HTML-документа, таблицы стилей и JavaScript-файла. Чтобы добавить поддержку автономного режима в свое приложение, включите атрибут `manifest` в элемент `html`, как показано в листинге 10.2.

Листинг 10.2. Добавление атрибута `manifest` в элемент `html`

```
<!DOCTYPE html>
<html manifest="application.manifest">
  .
  .
  .
</html>
```

Наряду с HTML-документом предоставьте файл манифеста и укажите в нем, какие ресурсы подлежат кешированию. Примерное содержимое файла манифеста представлено в листинге 10.3.

Листинг 10.3. Примерное содержимое манифеста кеша

```
CACHE MANIFEST
example.html
example.js
example.css
example.gif
```

Переход в автономный режим

Чтобы приложение могло распознавать наличие или отсутствие подключения к сети, в HTML5-совместимых браузерах предусмотрены дополнительные события.

Приложения могут иметь различные модели поведения в онлайновом (подключенном) и офлайновом (автономном) режимах работы. Включение некоторых дополнительных элементов в объект `window.navigator` упрощает решение этой задачи. Одним из таких элементов является `navigator.onLine` — булево свойство, указывающее, считает ли браузер, что он находится в онлайновом режиме. Разумеется, это вовсе не означает, что если свойство `onLine` равно `true`, что ваш компьютер может в любой момент беспрепятственно подключиться к серверам, с которыми должно связываться ваше приложение. С другой стороны, значение `false` означает, что браузер даже не будет пытаться подключиться к сети. В листинге 10.4 показано, как проверить, в каком режиме находится страница — онлайновом или офлайновом. Этот код будет работать даже в Internet Explorer.

Листинг 10.4. Проверка сетевого статуса

```
// Проверка подключения к сети при загрузке страницы
function loadDemo() {
  if (navigator.onLine) {
    log("Online");
  } else {
    log("Offline");
  }
}

// Добавить обработчики событий, извещающие об изменении
// сетевого статуса
window.addEventListener("online", function(e) {
  log("Online");
}, true);

window.addEventListener("offline", function(e) {
  log("Offline");
}, true);
```

Файлы манифеста

Автономные приложения включают в себя манифест, содержащий список ресурсов, которые браузер будет кешировать для автономного использования. Файлы манифеста имеют MIME-тип `text/cache-manifest`. Модуль `SimpleHTTPServer` в стандартной библиотеке Python обслуживает файлы, имеющие расширение `.manifest`, с заголовком `Content-type: text/cache-manifest`. Чтобы сконфигурировать настройки, откройте файл `PYTHON_HOME/Lib/mimetypes.py` и добавьте в него следующую строку:

```
'.manifest'     : 'text/cache-manifest manifest',
```

Для других серверов может потребоваться дополнительное конфигурирование. Например, для HTTP-сервера Apache следует обновить файл `mime.types` , находящийся в папке `conf`, добавив следующую строку:

```
text/cache-manifest manifest
```

Манифест состоит из набора текстовых строк и начинается со строки `CACHE MANIFEST` (листинг 10.5). Строки могут заканчиваться символами `CR`, `LF` или `CRLF` — формат достаточно гибок, но текст должен вводиться в кодировке UTF-8, типичной для большинства текстовых редакторов.

Листинг 10.5. Примерный файл манифеста, включающий все возможные разделы

```
# Строки комментариев начинаются с символа "решетки"
CACHE MANIFEST
# Кешируемые файлы
about.html
html5.css
index.html
happy-trails-rc.gif
lake-tahoe.JPG

#Страница signup не кешируется
NETWORK
signup.html

FALLBACK
signup.html      offline.html
/app/ajax/       default.html
```

Рассмотрим каждый раздел манифеста по отдельности.

Если не указан никакой заголовок, то по умолчанию используется раздел CACHE MANIFEST. Следующий простой манифест говорит о необходимости кеширования двух файлов.

```
CACHE MANIFEST
application.js
style.css
```

Включив файл в раздел CACHE MANIFEST, вы тем самым даете браузеру указание обслуживать этот файл из кеша приложения, даже если приложение работает в онлайновом режиме. Указывать здесь основной HTML-ресурс приложения излишне. HTML-документ, в котором первоначально задается файл манифеста, включается сюда неявно. Но если вы хотите кешировать несколько HTML-документов или если требуется, чтобы несколько HTML-документов играли роль возможных точек входа для кешируемого приложения, все они должны быть перечислены в файле манифеста кеша.

Записи FALLBACK позволяют предоставить пути к альтернативным ресурсам, если некоторые ресурсы не удается извлечь. В соответствии с манифестом из листинга 10.5, безрезультатные запросы к каталогу /app/ajax или подчиненным каталогам, пути к которым начинаются с /app/ajax, будут перенаправляться на файл default.html, если пути /app/ajax/* оказываются недоступными.

Записи NETWORK обозначают ресурсы, которые всегда должны извлекаться из источников, находящихся в сети, причем даже в тех случаях, когда пути запроса соответствует уже кешированный ресурс.

Объект applicationCache

Объект applicationCache реализует интерфейс для работы с кешем приложения. Новый объект window.applicationCache запускает несколько событий, связанных с состояниями кеша. Номер текущего состояния хранится в целочисленном свойстве window.applicationCache.status. Кеш приложения может находиться в одном из шести состояний, перечисленных в табл. 10.2.

Пока что манифесты кеша указываются очень редко и страницы не кешируются. Типичным состоянием для приложения с манифестом кеша является IDLE. В этом состоянии все ресурсы загружены браузером и никакая загрузка обновлений в это время не происходит. Кеш переходит в состояние OBSOLETE, если в какой-

то момент времени он существовал, но в данный момент манифест по каким-то причинам отсутствует. В программном интерфейсе некоторым из этих состояний соответствуют события (и атрибуты функций обратного вызова). Например, если переход кеша в состояние IDLE произошел после обновления, то запускается событие CACHED. В это время приложение может вывести уведомление о том, что пользователь может отключиться от сети и продолжит работу с приложением в автономном режиме. Некоторые типичные события и связанные с ними состояния кеша перечислены в табл. 10.3.

Таблица 10.2. Шесть состояний кеша приложения

Целочисленное свойство	Состояние кеша
0	UNCACHED
1	IDLE
2	CHECKING
3	DOWNLOADING
4	UPDATEREADY
5	OBSOLETE

Таблица 10.3. Типичные события и соответствующие им состояния кеша приложения

Событие	Состояние кеша
onchecking	CHECKING
ondownloading	DOWNLOADING
onupdateready	UPDATEREADY
onobsolete	OBSOLETE
oncached	IDLE

Кроме того, существуют события, указывающие на ход процесса обновления, отсутствие доступных обновлений и возникновение ошибки:

- onerror;
- onnoupdate;
- onprogress.

Объект window.applicationCache имеет метод update(), который запрашивает обновление кеша браузером. Этот процесс включает в себя проверку наличия новой версии файла манифеста и загрузку новых ресурсов, если это необходимо. В случае отсутствия или устаревания кеша генерируется ошибка.

Создание автономного веб-приложения

В этом примере приложения мы будем отслеживать местоположение участника соревнований в процессе его перемещения по маршруту (в условиях неустойчивой связи или даже отсутствия соединения). Например, Питер отправился на пробежку, имея при себе мобильный телефон с геолокационными возможностями HTML5 и веб-браузером HTML5, но сигнал в лесу около его дома не везде достаточно сильный. Питер хочет использовать это приложение для отслеживания своего местоположения и записи соответствующих данных, даже если соединение с Интернетом отсутствует.

В автономном режиме функции Geolocation API будут работать на устройствах с аппаратно реализованными средствами геолокации (например, GPS-навигаторы), но, очевидно, не на устройствах, использующих IP-геолокацию. Чтобы IP-геолокация могла работать, ей требуется подключение к Интернету для преобразования IP-адреса в коор-

динаты. Однако автономные приложения всегда могут получить доступ к хранилищу данных на локальном компьютере с помощью интерфейса локального хранилища или Web SQL Database. Пример работы этого приложения показан на рис. 10.2.

Рис. 10.2. Приложение в действии

Для выполнения этого приложения потребуется веб-сервер, обслуживающий статические ресурсы. Не забудьте, что файл манифеста должен обслуживаться с MIME-типом содержимого `text/cache-manifest`. Ели ваш браузер поддерживает кеш приложений, но файл манифеста обслуживается с некорректным типом содержимого, то это, скорее всего, приведет к возникновению ошибки.

Чтобы запустить данное приложение с полной функциональностью, потребуется сервер, который может получать геолокационные данные. Серверная часть, дополняющая этот пример, предположительно должна сохранять, анализировать и делать доступными соответствующие данные. Она может обслуживаться в виде статического приложения как тем же, так и другим источником.

На рис. 10.3 представлен пример приложения, выполняющегося в автономном режиме в Firefox. Файлы приложения для этого примера находятся на сайте книги в папке `code/offline`.

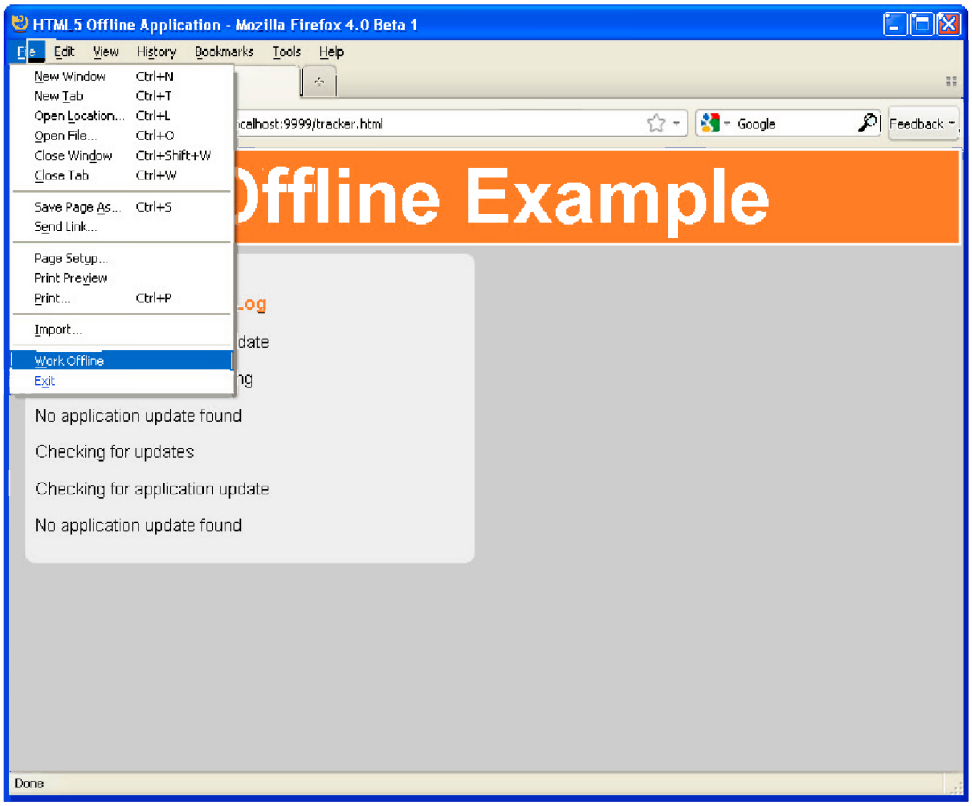


Рис. 10.3. Приложение, работающее в автономном режиме

Создание файла манифеста для ресурсов приложения

Прежде всего, создайте файл `tracker.manifest`, содержимое которого приведено ниже. Здесь будут перечислены файлы, входящие в состав данного приложения.

```
CACHE MANIFEST
# JavaScript-сценарии
./offline.js
#./tracker.js
./log.js

# таблицы стилей
./html5.css

# изображения
```

Создание HTML-структуры и CSS-файла для пользовательского интерфейса

Ниже представлена базовая структура пользовательского интерфейса примера. Файлы `tracker.html` и `html5.css` будут кешироваться, поэтому приложение использует кеш.

```
<!DOCTYPE html>
<html lang="en" manifest="tracker.manifest">
<head>
    <title>HTML5 Offline Application</title>
    <script src="log.js"></script>
    <script src="offline.js"></script>
    <script src="tracker.js"></script>
    <link rel="stylesheet" href="html5.css">
</head>
<body>
    <header>
      <h1>Offline Example</h1>
    </header>

    <section>
      <article>
        <button id="installButton">Check for Updates</button>
        <h3>Log</h3>
        <div id="info">
        </div>
      </article>
    </section>
</body>
</html>
```

По поводу содержимого этого HTML-документа можно сделать некоторые замечания относительно средств, обеспечивающих автономную работу приложения. Первое из них касается атрибута `manifest` элемента `html`. В большинстве HTML-примеров, приведенных в этой книге, элемент `<html>` опускался, поскольку он не является обязательным в HTML5, однако возможность использования автономного кеша зависит от указания файла манифеста.

Второе замечание касается кнопки. С помощью этой кнопки пользователь может управлять установкой приложения для работы в автономном режиме.

Создание JavaScript-сценария для автономного режима

JavaScript-сценарий для этого примера содержится в нескольких `.js`-файлах, включаемых дескрипторами `<script>`. Эти сценарии кешируются вместе с файлами HTML и CSS.

```
<offline.js>
/*
 * Занесение в журнал записей о событиях, запускаемых
 * объектом window.applicationCache
 */
window.applicationCache.onchecking = function(e) {
    log("Checking for application update");
}

window.applicationCache.onnoupdate = function(e) {
    log("No application update found");
}

window.applicationCache.onupdateready = function(e) {
    log("Application update ready");
}

window.applicationCache.onobsolete = function(e) {
```

```
    log("Application obsolete");
}

window.applicationCache.ondownloading = function(e) {
    log("Downloading application update");
}

window.applicationCache.oncached = function(e) {
    log("Application cached");
}

window.applicationCache.onerror = function(e) {
    log("Application cache error");
}

window.addEventListener("online", function(e) {
    log("Online");
}, true);

window.addEventListener("offline", function(e) {
    log("Offline");
}, true);


/*
 * Преобразование кодов состояния applicationCache в сообщения
 */
showCacheStatus = function(n) {
    statusMessages = ["Uncached","Idle","Checking","Downloading",
                      "Update Ready","Obsolete"];
    return statusMessages[n];
}

install = function() {
    log("Checking for updates");
    try {
        window.applicationCache.update();
    } catch (e) {
        applicationCache.onerror();
    }
}

onload = function(e) {
    // Убедиться в предоставлении браузером необходимой поддержки
    if (!window.applicationCache) {
        log("HTML5 Offline Applications are not supported in your
            browser.");
        return;
    }

    if (!navigator.geolocation) {
        log("HTML5 Geolocation is not supported in your browser.");
        return;
    }

    if (!window.localStorage) {
        log("HTML5 Local Storage not supported in your browser.");
        return;
    }
```

```
    log("Initial cache status: " +
        showCacheStatus(window.applicationCache.status));
    document.getElementById("installButton").onclick = checkFor;
}

<log.js>
log = function() {
    var p = document.createElement("p");
    var message = Array.prototype.join.call(arguments, " ");
    p.innerHTML = message;
    document.getElementById("info").appendChild(p);
}
```

Проверка поддержки кеша приложения

В дополнение к кешу автономного приложения в этом примере используются геолокация и локальное хранилище. Мы убеждаемся в том, что браузер поддерживает все три средства при загрузке страницы.

```
onload = function(e) {
    // Убедиться в предоставлении браузером необходимой поддержки
    if (!window.applicationCache) {
        log("HTML5 Offline Applications are not supported in your
            browser.");
        return;
    }

    if (!navigator.geolocation) {
        log("HTML5 Geolocation is not supported in your browser.");
        return;
    }

    if (!window.localStorage) {
        log("HTML5 Local Storage not supported in your browser.");
        return;
    }

    if (!window.WebSocket) {
        log("HTML5 WebSocket is not supported in your browser.");
        return;
    }
    log("Initial cache status: " +
        showCacheStatus(window.applicationCache.status));
    document.getElementById("installButton").onclick = install;
}
```

Добавление обработчика щелчка на кнопке обновления

Далее мы добавляем обработчик, обновляющий кеш приложения.

```
install = function() {
    log("Checking for updates");
    try {
        window.applicationCache.update();
    } catch (e) {
        applicationCache.onerror();
    }
}
```

Щелчок на кнопке Click for Update (Щелкните для обновления) запускает проверку обновлений кеша и, если это требуется, загрузку всех необходимых ресурсов. По окончании загрузки всех доступных обновлений выводится сообщение. Это говорит о том, что приложение успешно установлено и может выполняться в автономном режиме.

Добавление кода для отслеживания геолокационных координат

Приведенный ниже код базируется на аналогичном коде из главы 4 и содержится в JavaScript-файле `tracker.js`.

```
/*
 * Отслеживает текущее местоположение и создает
 * соответствующую запись
 */
var handlePositionUpdate = function(e) {
    var latitude = e.coords.latitude;
    var longitude = e.coords.longitude;
    log("Position update:", latitude, longitude);
    if(navigator.onLine) {
        uploadLocations(latitude, longitude);
    }
    storeLocation(latitude, longitude);
}

var handlePositionError = function(e) {
    log("Position error");
}

var uploadLocations = function(latitude, longitude) {
    var request = new XMLHttpRequest();
    request.open("POST",
                 "http://geodata.example.net:8000/geoupload",
                 true);
    request.send(localStorage.locations);
}

var geolocationConfig = {"maximumAge":20000};

navigator.geolocation.watchPosition(handlePositionUpdate,
                                    handlePositionError,
                                    geolocationConfig);
```

Добавление кода для работы с хранилищем

Затем мы добавляем код, записывающий обновления в хранилище `localStorage` при работе в автономном режиме.

```
var storeLocation = function(latitude, longitude) {
    // Загрузить список загруженных координат местоположения
    var locations = JSON.parse(localStorage.locations || "[]");
    // Добавить запись о местоположении
    locations.push({"latitude" : latitude,
                    "longitude" : longitude});
    // Сохранить новый список координат местоположения
    localStorage.locations = JSON.stringify(locations);
}
```

В этом приложении координаты сохраняются с использованием локального хранилища HTML5, описанного в главе 9. Локальное хранилище естественным образом согласуется с возможностью работать в автономном режиме, поскольку оно позволяет локально сохранять данные в браузере. Эти данные будут доступны в будущих сеансах. При восстановлении подключения к сети приложение сможет синхронизовать данные с сервером.

Дополнительным преимуществом использования хранилища в этом приложении является возможность восстановления после неудачных запросов выгрузки. Если приложение сталкивается с каким-либо сетевым сбоем или закрывается (в результате действий пользователя, сбоя браузера или операционной системы, а также в результате смены страниц), то данные сохраняются для будущей передачи.

Добавление обработчика событий перехода в автономный режим

Каждый раз, когда запускается обработчик обновлений, он проверяет состояние подключения. Если приложение работает в автономном режиме, оно просто сохраняет координаты. Когда соединение с сетью восстанавливается, приложение может обновить пользовательский интерфейс для отражения этого факта и выгрузки данных, сохраненных при работе в онлайновом режиме.

```
window.addEventListener("online", function(e) {
    log("Online");
}, true);

window.addEventListener("offline", function(e) {
    log("Offline");
}, true);
```

Состояние подключения может измениться в то время, когда приложение фактически не выполняется. Например, пользователь мог закрыть браузер, обновить страницу или перейти на другой сайт. Для обработки этих случаев в нашем автономном приложении при каждой загрузке страницы осуществляется проверка того, вернулось ли оно в онлайновый режим. Если это так, то приложение пытается синхронизировать данные с удаленным сервером.

```
// Если в данный момент браузер работает в автономном
// режиме, синхронизировать данные с сервером
if(navigator.onLine) {
    uploadLocations();
}
```

Резюме

В этой главе были продемонстрированы возможности автономных веб-приложений HTML5, которые могут использоваться даже при отсутствии подключения к Интернету. Можно добиться того, чтобы все ваши файлы кешировались, указав эти файлы в качестве компонентов веб-приложения в файле манифеста кеша и используя ссылки на них на основной HTML-странице приложения. Добавив обработчики событий для отслеживания состояния подключения, можно заставить свой сайт работать по-разному, в зависимости от того, доступно подключение к Интернету или нет.

В последней главе мы обсудим будущие перспективы программирования с использованием стандарта HTML5.

Глава 11

Будущее HTML5

Мы уже познакомились с целым арсеналом мощных возможностей, предоставляемых HTML5. Мы также обсудили историю развития HTML5 и его новую парадигму, делающую излишним использование подключаемых модулей. В этой главе мы попробуем заглянуть в будущее и обсудим те средства, которые пока еще находятся в стадии разработки, но их приход сулит многообещающие перспективы.

Поддержка HTML5 браузерами

HTML5 становится все популярнее с выходом очередных версий браузеров. Некоторые из рассмотренных нами средств фактически уже поставлялись вместе с браузерами, пока писалась эта книга. Никто не осмелится отрицать, что внедрение HTML5 в браузеры идет ускоренными темпами.

Действуя в несвойственной ей манере, компания Microsoft уже опубликовала несколько предварительных обзоров, посвященных следующей модели своего браузера. Internet Explorer 9 (IE9) будет поддерживать некоторые важные возможности HTML5 и улучшения, направленные на ускорение работы приложений. Когда IE9 появится на рынке[1], базовые средства современных браузеров будут охватывать значительную часть предусмотренных в HTML5 возможностей, касающихся работы с мультимедиа, хранения данных и обмена сообщениями.

Сегодня многие разработчики по-прежнему стараются придерживаться принципа обратной совместимости, стремясь к тому, чтобы их приложения могли работать даже с устаревшими моделями браузеров. Internet Explorer 6 оказался самым "стойким" из всех старых браузеров, которые широко используются по состоянию на 2010 год. Но даже для IE6 срок жизни ограничен, поскольку теперь все труднее подобрать операционную систему, которая бы его поддерживала. Пройдет время, и число компьютеров, на которых для работы с Интернетом используется IE6, сведется почти к нулю. Все больше и больше пользователей Internet Explorer обновляют браузер, устанавливая самые последние версии, что ведет к развитию Интернета. Устаревшие браузеры, заслуживающие нареканий, будут существовать всегда, но уровень требований постоянно повышается: на момент написания книги рыночная доля Internet Explorer 6 составляла 20% и продолжала падать. Большинство пользователей, обновляющих свои браузеры, сразу заменяют их современными версиями. Со временем минимальный уровень обязательной поддержки будет включать в себя сочетание средств HTML5 Video, Canvas, WebSocket и всего того, что вам пока что, возможно, приходится эмулировать, чтобы удовлетворить запросы самой широкой аудитории.

---

[1] Бета-версия веб-браузера Internet Explorer 9 была представлена компанией Microsoft в середине сентября 2010 г. — *Примеч. ред.*

В этой книге рассматривались те возможности и средства, которые могут считаться в значительной мере стабильными и поставляются со многими браузерами. Существуют также другие расширения HTML5 и программные интерфейсы, разработка которых началась совсем недавно. Эта глава познакомит вас с теми из них, появления которых следует ожидать в ближайшее время. Некоторые из них все еще находятся на ранней экспериментальной стадии, другие вскоре могут быть оформлены в качестве стандартов и станут широко доступными, подвергнувшись лишь незначительным изменениям по сравнению с их нынешним состоянием.

HTML развивается

В этом разделе мы рассмотрим некоторые из интересных средств, появления которых в браузерах можно ожидать в ближайшем будущем. Не исключено, что это произойдет еще до того, как наступит 2022 год. По всей видимости, эти возможности не будут формализованы в виде новой спецификации HTML6. Группа WHATWG дала понять, что направление будущих разработок будет иметь простое название — "HTML". Разработка стандарта будет вестись по принципу постепенного наращивания возможностей, причем конкретные средства и их спецификации будут развиваться самостоятельно, а не в соответствии с неким единым консолидированным планом. Новые средства будут внедряться в браузеры по мере достижения согласия между производителями браузеров и, возможно, станут широко доступными еще до того, как HTML5 оформится в виде законченной рекомендации. Сообщество, ответственное за дальнейшее развитие Интернета, прилагает все усилия к тому, чтобы развивающаяся платформа удовлетворяла потребности как разработчиков, так и пользователей.

WebGL

WebGL — это программный интерфейс, предназначенный для работы с трехмерной графикой в Интернете. Так уж исторически сложилось, что сразу несколько производителей браузеров, включая Mozilla, Opera и Google, начали независимо разрабатывать экспериментальные 3D-библиотеки для JavaScript. На сегодняшний день WebGL уверенно движется в направлении стандартизации и широкой доступности в HTML5-совместимых браузерах. В процессе стандартизации принимают участие производители браузеров и The Khronos Group — организация, ответственная за сопровождение OpenGL, кроссплатформенного стандарта 3D-графики, созданного в 1992 году. В настоящее время OpenGL, текущей версией которого является 4.0, широко используется в играх и системах автоматизированного проектирования (САПР), являясь одновременно аналогом и конкурентом библиотеки Direct3D корпорации Microsoft.

Как было показано в главе 2, двухмерный графический контекст создается путем вызова функции `getContext("2d")` с помощью элемента `canvas`. Стоит ли удивляться тому, что дверь остается распахнутой и для других графических контекстов. WebGL также использует элемент `canvas`, но уже через 3D-контекст. В текущих реализациях в аргументах функции `getContext()` используются экспериментальные префиксы поставщиков (`moz-webgl`, `webkit-3d` и т.п.). Например, в сборке Firefox с поддержкой WebGL можно получить 3D-контекст путем вызова `getContext("moz-webgl")` для элемента `canvas`. Интерфейс объекта, возвращаемого этим вызовом функции `getContext()`, отличается от его 2D-эквивалента, поскольку предоставляет не графические операции, а привязки OpenGL. Вместо того чтобы вызывать функции для рисования линий и заливки фигур, WebGL-версия контекста холста работает с буферами текстур и вершин.

HTML в трех измерениях

Подобно другим компонентам HTML5, WebGL — это неотъемлемая часть веб-платформы. Поскольку WebGL визуализируется в элементе `canvas`, он является частью документа. 3D-элементы `canvas` можно позиционировать и трансформировать на странице точно так же, как это делается с изображениями и 2D-элементами. По сути, вам доступны те же операции, что и для любого элемента `canvas`, включая наложение текста и видео, а также анимацию. Комбинация 3D-холстов с остальными элементами документа значительно упростит разработку индикаторных панелей и смешанных 2D- и 3D-интерфейсов по сравнению с чистыми 3D-технологиями отображения. Представьте, например, что у вас появляется возможность взять за основу 3D-сцену и наложить поверх нее простой пользовательский веб-интерфейс, используя HTML-разметку. В отличие от специально создаваемых меню и элементов управления, которые можно встретить во многих OpenGL-приложениях, программное обеспечение на основе WebGL будет включать в себя уже готовые элементы управления форм HTML5 привлекательного вида.

Хорошим дополнением к WebGL будет и существующая сетевая архитектура Интернета. Приложения на основе WebGL смогут извлекать из различных сайтов такие ресурсы, как текстуры и модели. Один из наглядных примеров представлен на рис. 11.1. Недавно Google перенесла классическую 3D-игру Quake II в Интернет, используя технологии HTML5 WebSocket и WebGL, предоставив возможность соревноваться сразу нескольким игрокам. Логика и графика игры были реализованы средствами JavaScript с использованием программного интерфейса WebGL для выполнения визуализации. Связь с сервером для координации действий игроков была обеспечена за счет использования WebSocket-соединений с сохранением состояния.

Рис. 11.1. Игра Quake II в браузере

3D-шейдеры

WebGL реализуется в виде JavaScript-привязки к библиотеке OpenGL ES 2 и поэтому использует программируемый графический конвейер, стандартизованный

в OpenGL, включая шейдеры. Шейдер — это графическая программа, позволяющая создавать чрезвычайно гибкие эффекты визуализации, которые применяются в трехмерных сценах для повышения реалистичности изображения. Шейдеры WebGL написаны на языке GL Shading Language (GLSL). Таким образом, в инструментарий Интернета добавляется еще один специализированный язык. Приложение HTML5, поддерживающее WebGL, включает в себя следующие компоненты: HTML — структурирование документа, CSS — стилевое оформление, JavaScript — реализация программной логики и GLSL — создание шейдеров. Разработчики, знакомые с OpenGL, могут применять имеющиеся навыки, работая с аналогичным интерфейсом в интернет-среде.

По всей видимости, WebGL послужит фундаментом для трехмерной графики в Интернете. Подобно библиотекам JavaScript, абстрагирующим интерфейс DOM, существуют библиотеки, обеспечивающие дополнительную функциональность поверх WebGL. В настоящее время разрабатываются библиотеки для графов сцен, 3D-файлов различного формата (например, COLLADA) и даже целых движков для разработки игр. На рис. 11.2 представлен Shader Toy — инструментальный шейдерный набор для WebGL, укомплектованный шейдерами группы Rgba Demogroup, специализацией которой является разработка демонстрационных сцен. В частности, на рисунке представлены результаты работы шейдера Leizex, созданного этой группой. Можно ожидать, что появление новых библиотек высокоуровневого рендеринга вооружит в ближайшее время новичков веб-программирования мощными средствами разработки трехмерных сцен.

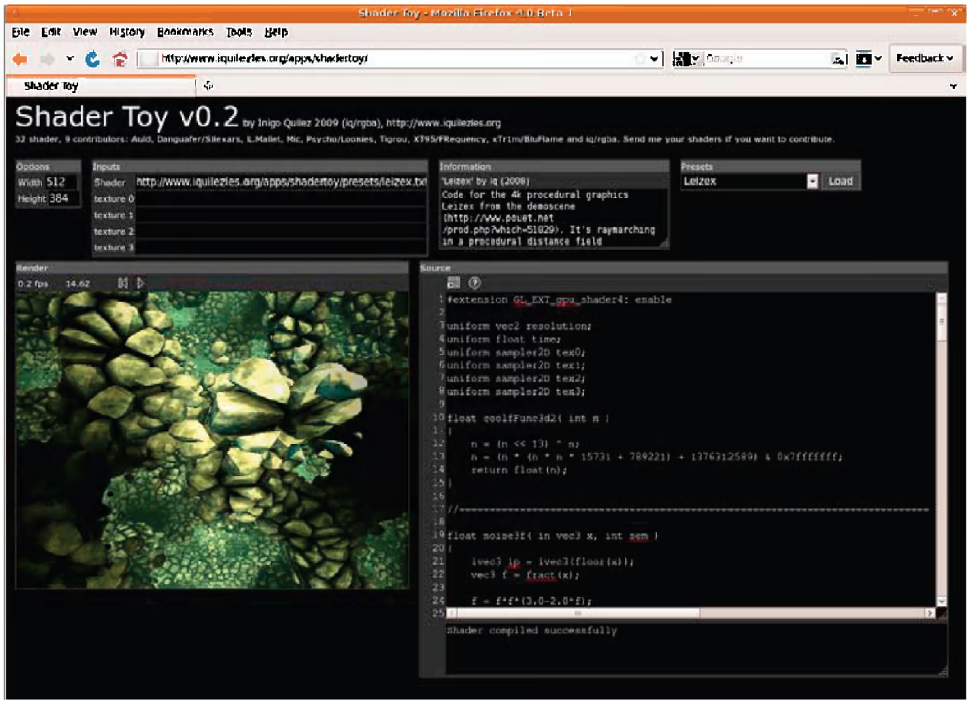


Рис. 11.2. Shader Toy — инструментальный набор шейдеров WebGL

Примечание

Разработка спецификации WebGL не только добавила возможности OpenGL в JavaDcript, но и повысила интерес к программным интерфейсам общего назначения, предназначенным для обработки двоичных данных. Быстрые 3D-интерфейсы нуждаются в буферизации двоичных данных. В связи с этим было подано новое предложение относительно объекта `TypedArray`, манипулирующего двоичными данными. Если это предложение будет принято, то, вероятно, объект `TypedArray` найдет свое применение в HTML5.

Устройства

Веб-приложения нуждаются в доступе к мультимедийному оборудованию — веб-камерам, микрофонам, подключаемым накопителям и т.п. Поэтому уже предложен элемент `device`, который должен предоставить веб-приложениям доступ к потокам данных, поступающим от подключенного оборудования. Разумеется, при этом возникают серьезные вопросы относительно безопасности личных данных, поскольку, например, не каждому сценарию следует безоговорочно предоставлять доступ к вашей веб-камере. Вероятно, в пользовательском интерфейсе появится элемент, запрашивающий у пользователя соответствующее разрешение, как в случае спецификаций Geolocation или Web Storage API, когда приложение запрашивает повышение полномочий. Очевидной областью применения для веб-камер являются видеоконференции, но для систем "компьютерного зрения" существует множество других заманчивых применений в веб-приложениях, включая виртуальную реальность и шлемы-дисплеи.

Программный интерфейс работы со звуком

Программные интерфейсы для работы со звуком будут играть по отношению к дескрипторам `<audio>` ту же роль, что и дескрипторы `<canvas>` по отношению к дескрипторам `<img>`. До появления элементов `canvas` изображения в Интернете в основном были недоступны для сценариев. Создание изображений и манипуляция ими должны были происходить на втором плане, то есть на сервере. Теперь же существуют инструменты для создания визуальной информации и манипулирования ею на основе элемента `canvas`. Точно так же интерфейсы работы со звуком позволят создавать музыку в приложениях HTML5. Это расширит возможности веб-приложений и приблизит нас к самодостаточному миру инструментов, позволяющих создавать мультимедийную информацию для Интернета средствами самого же Интернета. Представьте только, насколько удобно иметь возможность редактировать музыку, не покидая браузера.

Простое воспроизведение звуков обеспечивается элементом `audio`. В то же время любое приложение, осуществляющее манипуляции со звуком, а также его анализ или генерацию "на лету", нуждается в низкоуровневом программном интерфейсе. Преобразование текста в речь, синтезаторы, визуализация музыки — все это требует доступа к звуковым данным.

Используя элемент `device` и библиотеку функций для работы со звуком, вы сможете создавать приложения HTML5, позволяющие записывать и редактировать звук средствами веб-страницы. Аудиоклипы можно будет записывать в локальное хранилище браузера для повторного использования и сочетать со средствами редактирования на основе элемента `canvas`.

В настоящее время компания Mozilla разрабатывает экспериментальную реализацию интерфейса. Программный интерфейс Mozilla для работы со звуком может

быть использован в качестве отправной точки для обеспечения стандартных средств программирования звука, не зависящих от типа браузера.

Усовершенствованное видео

Уже имеющиеся возможности элемента `video` были подробно описаны ранее. В то время как в спецификации HTML5 по-прежнему отсутствует требование относительно обязательной поддержки видеокодеков, компания Google недавно выпустила WebM — высококачественный кодек для Интернета, не требующий лицензионных отчислений. Пока что WebM поддерживается не всеми производителями браузеров, но за этим форматом большое будущее.

Кроме того, существует большая потребность в улучшении имеющихся элементов управления для видео в HTML5. Чтобы полностью заменить визуализацию видео на основе плагинов, собственные элементы управления HTML должны будут ввести более совершенные программные интерфейсы для работы с потоковыми мультимедийными данными. Развития этих возможностей следует ожидать в улучшенных версиях медиадескрипторов. При все более широком распространении таких устройств, как Apple iPad и iPhone, в которых не используется Flash-технология, наблюдается сильное стремление донести видео в Интернете до широкой аудитории исключительно средствами HTML.

События сенсорных устройств

По мере того как доступ к Интернету все более смещается от настольных и переносных компьютеров к мобильным телефонам и планшетным устройствам, HTML стремится шагать в ногу с изменениями, затрагивающими интерактивное взаимодействие пользователя с приложениями. Когда компания Apple явила рынку iPhone, она ввела в его браузер набор специальных событий, предназначенных для обработки мультисенсорного ввода и поворота устройств. Хотя эти события еще не стандартизованы, они уже взяты на вооружение производителями других мобильных устройств. Изучая их, вы сможете уже сейчас оптимизировать свои веб-приложения для работы с наиболее популярными устройствами.

Ориентация

Простейшим событием для обработки на мобильном устройстве является событие *ориентации*. Его обработчик можно добавить в тело документа.

```
<body onorientationchange="rotateDisplay();">
```

В обработчике событий можно ссылаться на свойство `window.orientation`. Оно указывает на одно из четырех значений угла поворота, приведенных в табл. 11.1, которые отсчитываются относительно ориентации устройства при начальной загрузке страницы.

Таблица 11.1. Углы ориентации и их смысл

Угол ориентации	Смысл
0	Устройство сохраняет ориентацию, которую оно имело в момент начальной загрузки страницы
-90	Устройство повернуто на 90 градусов по часовой стрелке (вправо) относительно его положения в момент начальной загрузки страницы
180	Устройство повернуто на 180 градусов относительно его положения в момент начальной загрузки страницы
90	Устройство повернуто на 90 градусов против часовой стрелки (влево) относительно его положения в момент начальной загрузки страницы

Зная ориентацию устройства, можно соответствующим образом настроить содержимое на экране.

Жесты

Следующим типом событий, поддерживаемых мобильными устройствами, являются высокоуровневые события, называемые *жестами* (gesture). Жесты можно рассматривать как мультисенсорный ввод, управляющий изменениями размеров и поворотами. Обычно это происходит так: пользователь касается экрана одновременно двумя или несколькими пальцами и слегка надавливает на него или совершает пальцами вращательное движение. Вращение представляет поворот, тогда как усиление или ослабление давления на экран представляют соответственно уменьшение или увеличение размеров изображения. Для получения событий жестов приложение должно зарегистрировать один из обработчиков событий, представленных в табл. 11.2.

Таблица 11.2. Обработчики событий для жестов

Обработчик событий	Описание
`ongesturestart`	Пользователь коснулся экрана несколькими пальцами и приступил к выполнению жеста
`ongesturechange`	Пользователь выполняет жест с целью масштабирования или поворота изображения
`ongestureend`	Пользователь завершил выполнение жеста, убрав пальцы с экрана

В процессе выполнения жеста обработчик может проверять свойства `scale` и `rotation` событий и соответствующим образом обновлять изображение. Пример использования обработчика жестов представлен в листинге 11.1.

Листинг 11.1. Пример обработчика жестов

```
function gestureChange(event) {
   // Получить величину изменения масштаба в результате выполнения
   // жеста пользователем. Значение 1.0 представляет исходный
   // размер; меньшим значениям соответствует увеличение масштаба,
   // а большим -- уменьшение.
   var scale = event.scale;

   // Получить величину изменения угла поворота в результате
   // выполнения жеста пользователем. Угол поворота изменяется
   // в пределах от 0 до 360, причем положительным значениям
   // соответствует поворот по часовой стрелке, а отрицательным --
   // против.
   var rotation = event.rotation;

   // Обновить дисплей в соответствии с углом поворота.
}

// Зарегистрировать обработчик событий для узла документа
node.addEventListener("gesturechange", gestureChange, false);
```

События жестов особенно удобно использовать в тех случаях, когда требуется манипулировать объектами или изображениями. В частности, они могут применяться инструментами построения диаграмм или навигационными средствами.

Касания

В тех случаях, когда требуется низкоуровневое управление событиями устройств, всю необходимую информацию предоставят события *касаний* (touch). Различные события касаний представлены в табл. 11.3.

Таблица 11.3. События касаний

Обработчик событий	Описание
ontouchstart	Палец коснулся поверхности сенсорного устройства. При касании несколькими пальцами генерируются мультисенсорные события
ontouchmove	Один или несколько пальцев, помещенных на поверхность устройства, перемещаются, выполняя операцию перетаскивания
ontouchend	Один или несколько пальцев убраны с экрана устройства
ontouchcancel	Выполняемая касанием операция неожиданно прервана

В отличие от других событий мобильных устройств, события касания должны информировать о том, что существует одновременно несколько точек ввода (касание осуществляется сразу несколькими пальцами). Поэтому программный интерфейс для обработки касаний несколько более сложен, как показано в листинге 11.2.

Листинг 11.2. Функции для обработки касаний

```
function touchMove(event) {
   // Список touches содержит записи для каждого зафиксированного
   // в данный момент касания
   var touches = event.touches;

   // Список changedTouches содержит записи, соответствующие тем
   // касаниям, которые в данный момент испытали изменение:
   // нажатие, отпускание, перемещение точки касания.
   var changedTouches = event.changedTouches;

   // Список targetTouches содержит записи, соответствующие лишь
   // касаниям, относящимся к узлу, для которого зарегистрирован
   // данный обработчик.
   var targetTouches = event.targetTouches;

   // Получив события касаний, подлежащие отслеживанию, можно
   // ссылаться на большинство атрибутов, которые обычно
   // приходится получать из объектов других событий.
   var firstTouch = touches[0];
   var firstTouchX = firstTouch.pageX;
   var firstTouchY = firstTouch.pageY;
}

// Зарегистрировать один из обработчиков событий для нашего примера.
node.addEventListener("touchmove", touchMove, false);
```

Может случиться так, что собственные обработчики устройства мешают обработчикам касаний и жестов в вашем приложении. В подобных случаях используйте следующий вызов:

```
event.preventDefault();
```

Этот вызов отменяет предусмотренную по умолчанию обработку событий интерфейсом браузера и предоставляет вашему приложению возможность самостоятельно обрабатывать события. До тех пор пока события мобильных устройств не

будут стандартизованы, рекомендуется тщательно изучать документацию целевых устройств вашего приложения.

Пиринговые сети

В настоящее время наблюдается интенсивное внедрение передовых сетевых технологий в веб-приложения. Как в протоколе HTTP, так и в протоколе WebSocket существуют клиент (браузер или иной пользовательский агент) и сервер (URL-хост). Пиринговые (P2P) сети обеспечивают непосредственную связь клиентов между собой. Во многих случаях такая связь более эффективна, чем пересылка всех данных через сервер. Преимуществами такой связи являются снижение расходов на услуги хостинга и повышение производительности приложений. Пиринговые сети должны упростить создание быстрых многопользовательских игр и программ, предназначенных для организации коллективной работы.

Еще одной сферой непосредственного применения пиринговых сетей в сочетании с элементом `device` является эффективный видеочат в HTML5. В P2P-видеочате каждый из собеседников должен иметь возможность посылать данные непосредственно другому участнику разговора, без их маршрутизации через центральный сервер. P2P-видеочаты интенсивно используются приложениями наподобие Skype. Поскольку для потокового видео требуется широкая полоса пропускания, то, вероятно, без привлечения технологии пиринговой связи сама возможность существования подобных приложений была бы сомнительной.

Производители браузеров уже начали экспериментировать с пиринговыми соединениями. В качестве примера можно привести примененную в браузере Opera технологию Unite, которая обеспечивает хостинг упрощенного веб-сервера непосредственно в браузере. С помощью технологии Opera Unite пользователи могут создавать и предоставлять своим партнерам услуги чата, совместного использования файлов и коллективной работы над документами.

Конечно, для P2P-соединений в Интернете потребуется протокол, который учитывал бы посредников, обеспечивающих безопасность и передачу данных по сети, а также программный интерфейс, позволяющий создавать программы для работы с этим протоколом.

Главное направление

До сих пор мы концентрировали свое внимание на тех возможностях, которые HTML5 предоставляет разработчикам для создания мощных веб-приложений. Рассмотрим теперь ситуацию с другой точки зрения и оценим, что нового дает HTML5 *пользователям* этих приложений. Многие средства HTML5 позволяют устранять или снижать сложность сценариев и совершать действия, которые до этого были просто невозможны без привлечения плагинов. Например, с помощью элемента HTML5 `<video>` можно задавать элементы управления, режим автоматического воспроизведения, особенности буферизации и замещающее изображение без использования JavaScript-сценариев. Стандарт CSS3 позволил переместить анимацию и эффекты из сценариев в стили. Такой декларативный подход делает приложения более дружественными по отношению к пользователю и в конечном счете повышает производительность труда пользователей, которые ежедневно работают с вашими программными продуктами.

Вы уже видели, как инструменты разработки Firefox и WebKit предоставляют рабочую информацию об использовании хранилищ, а также чрезвычайно важную

информацию, касающуюся отладки, профилирования и вычислений в командной строке JavaScript. HTML-разработке теперь свойственны стремление к упрощению, декларативному коду и внедрению облегченного инструментария в браузеры и собственно веб-приложения.

Уверенность компании Google в стабильном развитии HTML5 настолько велика, что она объявила о грядущем выпуске операционной системы Google Chrome, базирующейся на браузере и медиаплеере. В операционной системе компании Google, выпуск которой ожидается в конце 2010 года, сделана попытка реализовать достаточно большую долю функциональности с помощью средств HTML для создания пользовательского интерфейса, в котором приложения доставляются с использованием стандартизованной веб-инфраструктуры.

Резюме

В этой главе мы вкратце ознакомились с ожидаемыми нововведениями HTML5, такими как новый элемент `device`, поддержка трехмерной графики, событий касаний и пиринговых сетей. Темпы разработки HTML5 не замедляются, и наблюдение за этим процессом доставляет немало удовольствия.

Мысленно обратимся к прошлому. Пусть те из вас, кто пользуется Интернетом или занимается разработкой в этой области на протяжении десяти и более лет, задумаются над тем, какой огромный путь проделала HTML-технология за последние несколько лет. Десять лет назад выражение "профессиональное программирование на HTML" подразумевало использование новых средств HTML 4. Передовые разработчики того времени только открывали для себя динамическое обновление страниц и запросы `XMLHttpRequest`. До появления термина "AJAX" должно было пройти еще несколько лет, хотя методики, охватываемые AJAX, уже начинали зарождаться. В основном приложения для браузеров создавались с целью управления фреймами и манипулирования картами изображений.

То, для чего ранее требовались функции, код которых растягивался на многие страницы, сегодня может быть выполнено средствами одной лишь разметки. Множество новых способов обмена сообщениями и взаимодействия стали доступными для тех, кто горит желанием загрузить один из многочисленных бесплатных HTML5-совместимых браузеров, открыть свой излюбленный текстовый редактор и окунуться в мир профессионального HTML5-программирования.

Мы надеемся, что проведенное нами исследование современного состояния веб-разработки доставило вам удовольствие и воодушевило на творческое применение новых знаний. Ждем появления новаций, которые вы создадите в предстоящее десятилетие с помощью HTML5, чтобы написать о них!

Предметный указатель